НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра прикладної математики

«На правах рукопису»
УДК 004.912

«До захисту допущено»
Завідувач кафедри
_____ І.А. Дичка
«___»_____2019 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 121 Програмна інженерія

на тему: «Модифікований метод визначення авторства тексту на основі ланцюгів Маркова»

Виконав:
студент ІІ курсу, групи КП-71мн
Замекула Олексій Ігорович _____

Керівник:
Доцент кафедри ПЗКС, к.т.н., доцент,
Заболотня Т.М. _____

Консультант з нормоконтролю:
Ст. викладач кафедри ПЗКС, к.т.н.
Онай М.В. _____

Рецензент:
к.т.н., доц, доц.ММСА ІПСА
Дідковська М.В. _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2019 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – другий (магістерський) за освітньо-науковою програмою

Спеціальність – 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Науковий керівник кафедри

_____ І.А. Дичка

«___»_____2017 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Замекулі Олексію Ігоровичу

1. Тема дисертації «Модифікований метод визначення авторства тексту на основі ланцюгів Маркова», науковий керівник дисертації Заболотня Тетяна Миколаївна, к.т.н., доцент, затверджені наказом по університету від «8» квітня 2019 р. №1075-С

2. Термін подання студентом дисертації «17» травня 2019 р.

3. Об'єкт дослідження: процеси автоматизованого аналізу текстових даних за різними ознаками, що характеризують авторський стиль, зокрема, буквосполучення та використання службових частин мови.

4. Предмет дослідження: методи, способи та алгоритми автоматизованого визначення авторства тексту.

5. Перелік завдань, які потрібно розробити:
   - провести аналіз методів визначення авторства тексту;
   - дослідити існуючі програмні засоби, що вирішують завдання визначення авторства;
   - розробити та дослідити модифікований метод на основі ланцбгів Маркова л=для визначення авторства тексту;
   - обґрунтувати вибір критеріїв оцінки ефективності модифікованого методу;
   - розробити програмний засіб, що реалізує модифікований метод;
   - проаналізувати ефективність модифікованого методу в порівнянні з класичним на основі визначених критеріїв ефективності.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу:
   - теоретичні аспекти побудови та оптимізації модифікованого методу визначення авторства тексту на основі ланцюгів Маркова;
   - схема роботи модифікованого методу визначення авторства тексту;
   - схема попередньої обробки текстових даних;
   - схема роботи модулю попередньої обробки;
   - архітектура системи, що реалізує модифікований метод визначення авторства;
   - результат порівняльного аналізу модифікованого та оригінального методу за критеріями точності та швидкодії.

7. Орієнтовний перелік публікацій:
   - Стаття “Модифікований метод визначення авторства тексту на основі ланцюгів Маркова”
   - Тези доповіді “Порівняння статистичних методів визначення авторства тексту на основі художніх творів, написаних українською мовою”

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент кафедри ПЗКС		

9. Дата видачі завдання «11» жовтня 2017 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Ґрунтовне ознайомлення з предметною галуззю	18.12.2017	
2.	Визначення структури магістерської дисертації; вивчення літератури, пошук додаткової літератури, патентний пошук	05.03.2018	
3.	Робота над першим розділом магістерської дисертації; проведення наукового дослідження	15.05.2018	
4.	Проведення наукового дослідження; робота над другим розділом магістерської дисертації; розроблення програмного забезпечення; підготовка матеріалів доповіді на конференції ПМК-2018	16.10.2018	
5.	Проведення наукового дослідження; робота над статтею за результатами наукового дослідження	17.12.2018	
6.	Проведення наукового дослідження; робота над третім розділом магістерської дисертації	05.03.2019	
7.	Завершення роботи над основною частиною магістерської дисертації; підготовка ілюстративного матеріалу	27.04.2019	
8.	Оформлення текстової і графічної частини магістерської дисертації	01.05.2019	

Студент О.І. Замекула

Науковий керівник дисертації Т.М. Заболотня

РЕФЕРАТ

Актуальність теми. Задача визначення авторства тексту тривалий час залишається актуальною у багатьох галузях діяльності суспільства. Прикладами застосування рішень даної задачі є визначення авторства у видавництві, освітній галузі тощо. На сьогоднішній день кількість текстової інформації, поданої в електронному вигляді, інтенсивно зростає, тож необхідним стає розроблення ефективних методів автоматизованого визначення авторства текстів. В даній магістерській дисертації питання визначення авторства тексту розглядається з метою пошуку більш ефективних за критерієм точності методів визначення авторського інваріанту.

Об'єктом дослідження в даній роботі є процеси автоматизованого аналізу текстових даних за різними ознаками, що характеризують авторський стиль, зокрема, буквосполучення, слова та використання службових частин мови.

Предметом дослідження є методи, способи та алгоритми автоматизованого визначення авторства тексту.

Мета дослідження полягає у підвищені точності визначення авторства тексту шляхом розроблення та реалізації модифікованого методу аналізу текстів на основі методу ланцюгів Маркова та відповідних програмних засобів.

Методи дослідження. В роботі використовуються методи комп'ютерної лінгвістики, статистичні методи та емпіричні методи.

Наукова новизна роботи полягає в наступному:

1. Запропоновано модифікований метод автоматизованого визначення авторства текстів на основі методу ланцюгів Маркова, який відрізняється від існуючих класичних методів підвищеною точністю визначення автора за рахунок використання методу пошуку авторського інваріанту на основі частотних характеристик тексту.

Практична цінність отриманих в роботі результатів полягає в тому, що запропонований модифікований метод автоматизованого визначення авторства тексту реалізований для визначення автора невідомого тексту.

Крім того в рамках даного дослідження розроблено програмне забезпечення для використання при подальшій роботі над цією тематикою.

Апробація роботи. Основні положення та результати роботи були представлены на науковій конференції магістрантів та аспірантів “Прикладна математика та комп’ютинг” ПМК-2018-2 та опубліковані у збірнику тез доповідей.

Структура та обсяг роботи. Магістерська дисертація складається з вступу, чотирьох розділів, висновків та додатків.

У вступі надано загальну характеристику роботи, виконано оцінку сучасного стану проблеми, обґрунтовано актуальність напрямку досліджень, сформульовано мету і задачі дослідження.

У першому розділі розглянуто задачу визначення авторства, виконано огляд існуючих методів та порівняння програмних комплексів, які реалізують ці методи.

У другому розділі розглянуто основні методи попередньої обробки текстової інформації, порівняно формальні методи визначення авторства а також характеристики для визначення авторського інваріанту, запропоновано модифікований метод на основі методу ланцюгів Маркова.

У третьому розділі описано засоби, які було використано для проектування та розробки програмного забезпечення, яке реалізує модифікований метод а також описано алгоритми даного програмного забезпечення.

У четвертому розділі наведений аналіз результатів роботи запропонованого методу, порівняння з існуючими методами.

У висновках проаналізовано отримані результати роботи.

У додатках наведено результати роботи модифікованого методу та лістинг коду розробленого програмного забезпечення.

ABSTRACT

Actuality of theme. The task of determining the authorship of the text for a long time remains relevant in many sectors of society. Examples of solutions to this problem are the definition of authorship in the publishing industry, the educational industry, etc. To date, the amount of textual information submitted in electronic form is intensively increasing, therefore, the development of effective methods for the automated determination of authorship of texts is becoming necessary. In this master's thesis, the question of determining the authorship of the text is considered in order to find more effective criteria for the accuracy of methods for determining the author's invariance.

The object of research in this paper is the processes of automated analysis of text data on various features that characterize the author's style, in particular, the letter combination, the words and the use of the service parts of the language.

The subject of the study is the methods, methods and algorithms of automated determination of authorship of the text.

The purpose of the study is to increase the accuracy of determining the authorship of the text by developing and implementing a modified method for analyzing texts based on the method of Markov chains and related software.

Research methods. Methods of computer linguistics, statistical methods and empirical methods are used in this work.

The scientific novelty of the work is as follows:

1. A modified method of automated determination of the authorship of texts based on the Markov chain method is proposed, which differs from the existing classical methods by the increased accuracy of the author's definition by using the method of searching the author's invariant on the basis of the frequency characteristics of the text.

The practical value of the results obtained in the work is that the proposed modified method of automated definition of the authorship of the text is implemented to identify the author of an unknown text.In addition, in the framework of this study, software was developed for use in further work on this topic.

Test work. The main provisions and results of work were presented at the scientific conference of masters and postgraduates "Applied Mathematics and Computer" PMK-2018-2 and published in the abstracts.

Structure and scope of work. The master's thesis consists of an introduction, four chapters, conclusions and appendices.

The introduction gives a general description of the work, evaluates the current state of the problem, substantiates the relevance of the research direction, formulates the purpose and objectives of the study.

In the first section the problem of determination of authorship is considered, a review of existing methods and comparison of software complexes that implement these methods are performed.

The second chapter deals with the basic methods of preliminary processing of textual information, the relatively formal methods of determining the authorship, as well as the characteristics for determining the author's invariant, proposed a modified method based on the method of Markov chains.

The third section describes the tools that were used to design and develop software that implements the modified method and describes the algorithms of this software.

In the fourth section is an analysis of the results of the proposed method, comparison with existing methods.

The conclusions are analyzed the results of work.

The annexes present the results of the modified method and the listing of the code of the software developed.

ЗМІСТ

ВСТУП

Проблема визначення авторства тексту завжди була об'єктом дослідження у філології. У історії світової філологічної науки майже при самому її зародженні виникло питання про достовірність і підробленість письмового тексту, а також про прийоми і принципи визначення автора літературного твору.

Враховуючи вищевикладене, на цей час постає питання не тільки визначення авторства тексту, а і захисту авторських прав. Так, на міжнародному рівні в 1952 році у Женеві була прийнята Всесвітня конвенція про авторське право, головною метою якої стало створення процедур для захисту авторських прав як літературних, наукових , так і художніх творів. Законодавство поширюється на різні твори: музичні, письмові, живопис, графіку тощо, та ним визначено чіткий порядок та вимоги до оформлення, публікації творів, гарантуються конкретні права авторів (матеріальні та нематеріальні) на весь період їх охорони, саме : впродовж життя автора та до 25 років після смерті автора

На сьогоднішній день, з розвитком засобів комунікації та розповсюдження інформації за допомогою мережі Інтернет, проблема авторства тексту стоїть особливо гостро. Через стрімкий розвиток інформаційних технологій ми спостерігаємо експоненційне збільшення обсягів текстової інформації. Тож у таких умовах неможливо проводити аналіз тексту експертами.

Розроблення математичних методів дослідження текстів почалась ще на початку XX століття. Проте досі немає формалізованого методу дослідження тексту на авторство, який давав би стовідсотковий результат. Також завдання ускладнює розмаїття природніх мов, адже при дослідженні необхідно враховувати лексичні, синтаксичні та інші особливості кожної з них.

Отже, розроблення нових та удосконалення існуючих методів автоматизованого визначення авторства тексту, які дозволили б підвищити ефективність вирішення цієї задачі, є актуальним завданням сьогодення.

1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ЩОДО ВИЗНАЧЕННЯ АВТОРСТВА ТЕКСТУ

1.1. Загальний опис задачі визначення авторства тексту

Питання статистичного визначення автора літературного тексту постійно знаходяться в сфері інтересів як літературознавців, так і математиків. Практичну важливість мають аспекти історичні, правові, культурологічні. Викликає інтерес і саме дослідження творчої роботи мозку з точки зору алгоритмічності цього процесу. Також мають самостійну цінність і статистичні методи аналізу, розвинені стосовно таких багатовимірних за своїми атрибутами об'єктів, як літературні тексти, написані професійними письменниками. Останнє означає, що письменник в своїх творах дотримується певної манери письма, що і дозволяє використовувати статистичні методи для визначення належних йому текстів.

Для визначення справжнього автора тексту часто доводиться звертатися до експертів, які можуть ідентифікувати автора невідомого тексту або визначити належність твору іншому автору за допомогою характерних мовних особливостей і різних стилістичних прийомів.

Попри це, у сучасних реаліях такі задачі постають у дуже великим масштабах, що не дозволяє робити експертну оцінку достатньо точно та швидко для потреб людства. Тож з розвитком ЕОМ дуже активно стали розроблятись методи автоматизації задач визначення авторства тексту. Такі методи, засновані на математиці, називають формальними. На сьогоднішній день для атрибуції текстів застосовуються підходи з теорії розпізнавання образів, математичної статистики та теорії ймовірностей, алгоритми нейронних мереж і кластерного аналізу та багато інших.

Атрибуція тексту – дослідження тексту з метою встановлення авторства або отримання будь-яких відомостей про автора і умови створення текстового

документа. Завдання атрибуції можна розділити на ідентифікаційні і діагностичні [1].

Ідентифікаційні завдання вирішуються з припущення, що автор тексту відомий. Такі завдання вирішують для перевірки, чи дійсно автором є певна людина, чи ні. Вирішення такої задачі цікавить передусім літературознавців, які прагнуть встановити дійсність належності твору певному автору. Також ця задача має вирішуватись для потреб виконання законодавчих актів, пов’язаних з авторськими правами.

Діагностичні завдання вирішуються з припущення, що автор тексту невідомий. Вони дозволяють визначити особистісні характеристики автора (освітній рівень, рідна мова, знання іноземних мов, походження, місце постійного проживання та ін.) і факт свідомого спотворення письмової мови. Вирішення цього завдання має дуже велике значення, як приклад, для істориків, які таким чином можуть співставити автора тексту з іншими авторами певного історичного проміжку, таким чином більш точно датувати текст.

Формальні методи частіше всього засновані на порівнянні обчислюваних характеристик текстів. У загальному випадку текст відображається в вектор обчислених для нього параметрів, кожен з яких об'єктивно характеризує певний набір особливостей тексту. Таким чином, текст графічно відображається в деяку точку n-мірного простору. При такій формалізації авторський стиль також може бути представлений у вигляді аналогічного вектору параметрів – цим вектором буде вектор текстів, написаних даним автором.

Таким чином, задача атрибуції зводиться до обчислення певних числових характеристик тексту, та представлення його у вигляді вектору, або точки n-мірного простору. За умови, що ми маємо оцінку деяких текстів цього ж автора, достатньо буде оцінити “відстань” між цією точкою і іншими

текстами. У цьому випадку “відстань” буде позначати відмінність між числовими характеристиками тексту, що аналізується, та тих, що вже було проаналізовано.

Використання таких методів дозволяє вирішувати обидві задачі атрибуції одними і тими ж методами, оскільки у пам’яті ЕОМ можна зберегти велику кількість проаналізованих текстів і аналізувати нові набагато швидше, ніж це може робити експерт.

Разом з тим, постають деякі труднощі. Першою проблемою є те, що для достатньої точності аналізу необхідна велика кількість текстів різних авторів, щоб характеристика їх стилю була якомога точнішою. Відповідно, якщо у базі авторів немає текстів автора, твір якого аналізується, дізнатись авторство буде неможливо.

Найбільшою ж проблемою постає вибір числових характеристик тексту. Однією з необхідних умов для вибору характеристики є її швидка обчислюваність. Для першої умови застосовують попередню обробку тексту, щоб прибрати компоненти, які не мають значення для аналізу. Ці методи можуть різнитися залежно від лексичних особливостей тієї чи іншої мови. Другою і основною умовою є те, щоб ця характеристика досить корректно характеризувала авторський стиль. Тобто, необхідно, щоб для різних авторів вона відрізнялась достатньо сильно, а також щоб її було важко свідомо контролювати. На сьогодні відомо багато характеристик, які застосовуються у формальних методах визначення авторства. Проте абсолютно точного формального методу аналізу тексту на авторства не існує.

Також перешкодою для алгоритмів визначення авторства може бути малий обсяг вхідного тексту [1, 2].

1.2. Огляд методів атрибуції тексту

Згадно з темою даної роботи основним методом, на основі якого будується новий метод, будемо вважати метод ланцюгів Маркова. Проте, проведемо аналіз інших методів атрибуції тексту задля виявлення їх спільних та відмінних рис. Це дасть можливість сформувати гіпотези щодо модифікації методу ланцюгів Маркова.

Методи атрибуції дозволяють досліджувати текст на п'яти рівнях: пунктуаційні, орфографічному, синтаксичному, лексико-фразеологічному, стилістичному.

Пунктуаційний рівень допомагає виявити особливості вживання автором знаків пунктуації, характерні помилки.

Орфографічний рівень виявляє характерні помилки в написанні слів.

Синтаксичний рівень дозволяє визначити особливості побудови речень, перевагу тих чи інших мовних конструкції, вживання часів, активного або пасивного стану дієслів, порядок слів, характерні синтаксичні помилки.

Лексико-фразеологічний рівень визначає словниковий запас автора, особливості використання слів і виразів, схильність до вживання рідкісних і іноземних слів, діалектизмів, архаїзмів, неологізмів, професіоналізмів, арготизмів, навички застосування фразеологізмів, прислів'їв, приказок, «крилатих виразів» і т.д.

Стилістичний рівень дозволяє визначити жанр, загальну структуру тексту, для літературних творів сюжет, характерні засоби (метафора, іронія, алегорія, гіпербола, порівняння), стилістичні фігури (градація, антитеза, риторичне питання і т.д.), інші характерні мовні прийоми.

Під «авторським стилем» зазвичай розуміються останні три рівня. Аналіз саме синтаксичного, лексико-фразеологічного та стилістичного рівнів становить найбільший інтерес і найбільшу складність [1].

Розглянемо детальніше формальні методи визначення авторства тексту.

В якості критерію близькості двох текстів вводиться так чи інакше обчислена «відстань» між відповідними векторами. У найпростішому випадку можна уявити набори параметрів як звичайні вектори в *n*-вимірному Декартовому просторі, що виходять з початку координат, і вважати відстанню між текстами звичайну Декартову відстань між кінцями відповідних їм векторів. Саме «відстань» є в підсумку інтегральною характеристикою відмінності текстів. Вона певним чином нормується, і тексти, для яких відстань велика, з високою ймовірністю належать різним авторам. Таким чином, щоб зіставити авторство двох текстів, досить обчислити для них параметри і визначити відстань. Щоб зіставити текст з автором, порівнюються вектори параметрів авторського стилю і даного тексту, тобто фактично знову порівнюються два тексти: текст зі свідомо відомим автором (еталонний текст) і текст, авторство якого потрібно встановити, підтвердити або спростувати (спірний текст). Для більшої точності у якості еталонного тексту зазвичай використовуються усереднені характеристики авторського стилю, які визначаються за результатами аналізу певного набору текстів для кожного автора.

Можна також скласти вектори формальних параметрів, що розрізняють не конкретних авторів (або їх групи), а виділяють певні характеристики авторів (наприклад, освітній рівень).

У більшості випадків в якості характеризуючих параметрів тексту вибираються його статистичні характеристики: кількість використання певних частин мови, деяких конкретних слів, знаків пунктуації, фразеологізмів, архаїзмів, рідкісних та іноземних слів, кількість і довжина речень (виміряна в словах, складах, знаках), обсяг словника, кількість повнозначних і службових слів, середня довжина речень, відношення кількості дієслів до загальної кількості слововживань в тексті і т.д. [1, 2].

Основна проблема формальних методів аналізу авторства, як було зазначено вище, полягає якраз у виборі параметрів. Як було відзначено А.А. Марковим [3], існує цілий ряд формальних статистичних характеристик текстів, непридатних для визначення авторства в силу одного з таких недоліків:

1. Відсутність стійкості.
2. Відсутність відрізняючої здатності.

Якими б не були параметри, завжди є ймовірність, що два або декілька учасників виявляться близькі за даними параметрами по причині випадкового збігу. Тому на практиці прийнято вважати достатнім, якщо параметр дозволяє впевнено розрізняти різні групи авторів.

Розглянемо існуючі методи атрибуції більш детально.

Графічно формальні статистичні методи визначення авторства тексту можна представити у вигляді схеми (рис. 1.1).

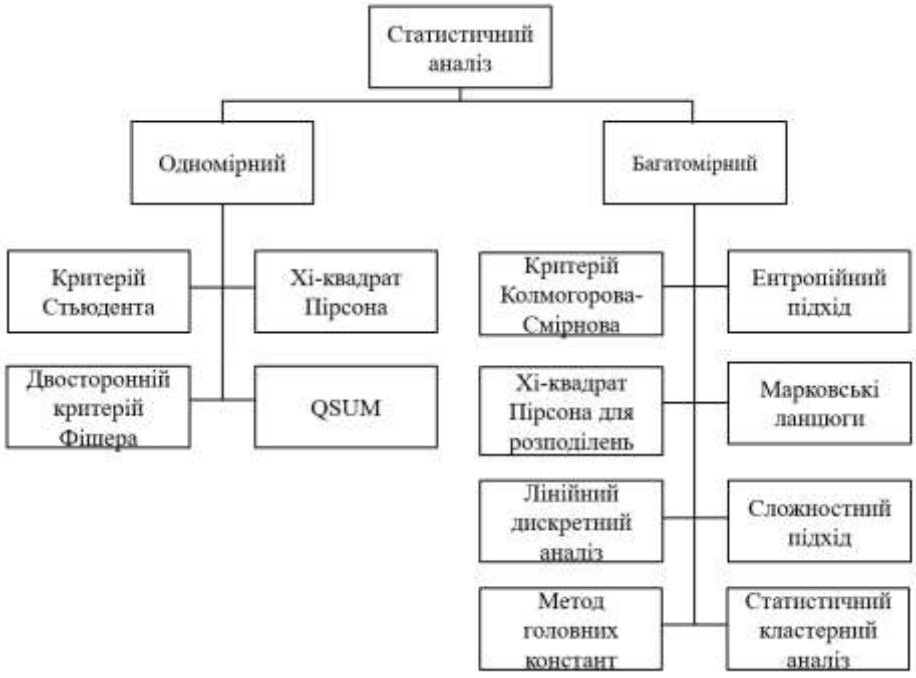


Рис. 1.1. Статистичні методи визначення авторства

Категорія одномірних статистичних методів характеризується вибором однієї змінної у якості критерію. Оскільки текст – досить складний об’єкт, який має багато характеристик, одновимірний аналіз не може забезпечити достатньої точності визначення автора.

На відміну від одновимірного аналізу – багатовимірний аналіз дозволяє аналізувати об’єкт одразу за багатьма характеристиками. У нашому випадку для точності визначення автора необхідно застосовувати саме багатовимірний аналіз, тож розглянемо детальніше цю групу.

1.2.1. Метод головних компонент (констант)

Метод головних компонент (МГК) застосовується для зниження розмірності простору спостережуваних векторів, не призводячи до істотної втрати інформативності. Передумовою МГК є нормальний закон розподілу багатовимірних векторів. У МГК лінійні комбінації випадкових величин визначаються характеристичними векторами коваріантної матриці. Головні компоненти є ортогональною системою координат, в якій дисперсії компонент характеризують їх статистичні властивості.

Однак метод не завжди ефективно знижує розмірність за заданих обмежень на точність. Прямі і площини не завжди забезпечують гарну апроксимацію. Наприклад, дані можуть з хорошою точністю дотримуватися якоїсь кривої, а ця крива може бути складно розташована у просторі даних. У цьому випадку метод головних компонент для прийнятної точності зажадає декількох компонент (замість однієї), або взагалі не дасть зниження розмірності за прийнятної точності. Для роботи з такими «кривими» головними компонентами винайдено метод головних многовидів [4] і різні версії нелінійного методу головних компонент [5, 6]. Найбільше неприємностей можуть спричинити дані складної топології. Для їх апроксимації також винайдено різні методи, наприклад самоорганізаційні

карти Кохонена, нейронний газ [7, 8] або топологічні граматики [9]. Якщо дані статистично породжені з розподілом, що дуже відрізняється від нормального, то для апроксимації розподілу корисно перейти від головних компонент до незалежних компонент [8], які вже не ортогональні у початковому скалярному добутку. Нарешті, для ізотропного розподілу (навіть нормального) замість еліпсоїда розсіювання отримуємо кулю, і зменшити розмірність методами апроксимації неможливо.

1.2.2. Кластерний аналіз

Термін кластерний аналіз (уперше ввів Tryon, 1939) насправді включає набір різних алгоритмів класифікації. Загальне питання, що ставиться дослідниками у багатьох областях, полягає в тому, як організувати дані спостережень в наочні структури, тобто розгорнути таксономію.

Фактично, кластерний аналіз є не стільки звичайним статистичним методом, скільки "набором" різних алгоритмів "розподілу об'єктів по кластерах". Існує точка зору, що на відміну від багатьох інших статистичних процедур, методи кластерного аналізу використовуються у більшості випадків тоді, коли відсутні яких-небудь апріорні гіпотези відносно класів.

Рішення задачі кластеризації принципове неоднозначне, і цьому є декілька причин:

- Не існує однозначно якнайкращого критерію якості кластеризації. Відомий цілий ряд евристичних критеріїв, а також ряд алгоритмів, що не мають чітко вираженого критерію, але здійснюють достатньо розумну кластеризацію «по побудові». Всі вони можуть давати різні результати.
- Число кластерів, як правило, невідоме заздалегідь і встановлюється відповідно до деякого суб'єктивного критерію.

– Результат кластеризації істотно залежить від метрики, вибір якої, як правило, також суб'єктивний і визначається експертом [10].

1.2.3. Критерій Колмогорова

Класичний критерій Колмогорова (іноді використовують Колмогорова-Смирнова) призначений для перевірки простих гіпотез про приналежність аналізованої вибірки деякому повністю відомому закону розподілу.

Нехай – вибірка незалежних однаково розподілених випадкових величин – емпірична функція розподілу, деяка "істинна" функція розподілу з відомими параметрами. Статистика критерію визначається виразом:

$$D_n = \sup|F_n(x) - F(x)|.$$

Позначимо через гіпотезу про те, що вибірка підкоряється розподілу. Тоді по теоремі Колмогорова при справедливості гіпотези, що перевіряється :

$$\forall t > 0: \lim_{n\to\infty} P\left(\sqrt{n}D_n \le t\right) = K(t) = \sum_{j=-\infty}^{+\infty}(-1)^{j_e - 2j^2t^2}.$$

Гіпотеза H0 відкидається, якщо статистика $\sqrt{n}D_n$ перевищує квантиль розподілення K_α заданого рівня значимости α, а інакше приймається [11, 12].

1.2.4. Хі-квадрат Пірсона

Хі-квадрат Пірсона один з найпопулярніших статистичних критеріїв для аналізу якісних даних (номінальних, порядкових, рангових), аналізу частот. Проте, як і у кожного статистичного критерію у хі-квадрата є свої власні правила застосування методу, його інтерпретації.

Хі-квадрат використовується передусім для аналізу таблиць зв'язаності (вид таблиці, яка враховує спільний вплив чинника на результат, дані в таблиці зв'язаності мають бути представлені у вигляді частоти номінальних даних або інтервалами, але не безперервними кількісними величинами). Варто відзначити, що при роботі із зв'язаними таблицями хі-квадрат часто є

підтримкою для аналізу впливу чинників ризику за допомогою розрахунку ризиків (абсолютний і відносний ризики) і відношення шансів. Таблиці зв'язаності можуть приймати різні форми.

Умови застосування статистичного критерію хі-квадрата Пірсона. Тип даних: параметри мають бути якісними цілісно чисельними частотами, виміряними в номінальній шкалі, порядковими. Бажано, щоб загальна кількість спостережень була більше 20, очікувана частота, що відповідає нульовій гіпотезі має бути більше 5, якщо очікуване явище набуває значення менше 5, то необхідно використати точний критерій Фішера [13, 14].

Основна формула для розрахунку хі-квадрата Пірсона:

$$x_n^2 = \sum_{i=1}^{n} \frac{(0_i - E_i)^2}{E_i}.$$

1.2.5. Метод Маркова

Ланцюгом Маркова називають таку послідовність випадкових подій, в якій вірогідність кожної події залежить тільки від стану, в якому процес знаходиться у нинішній момент і не залежить від більше ранніх станів. Кінцевий дискретний ланцюг визначається множиною станів $S = \{s1.,sn\}$, подією є перехід з одного стану в інший в результаті випадкового випробування вектором початкової вірогідності (початковим розподілом) $p\ (0) = \{p\ (0)\ (1).,p\ (0)\ (n)\}$, що визначає вірогідність $p\ (0)\ (i)$ того, що в початковий момент часу t = 0 процес знаходився в змозі s_i матрицею перехідної вірогідності $P = \{p_{ij}\}$, що характеризує вірогідність переходу процесу з поточним станом s_i в наступний стан s_j, при цьому сума вірогідності переходів з одного стану дорівнює 1:

$\sum j = 1 \ldots n, p_{ij} = 1$

Приклад матриці перехідної вірогідності з множиною станів $S = \{S1.,S5\}$, вектором початкової вірогідності $p\,(0) = \{1,0,0,0,0\}$:

$$P = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \end{array} \begin{array}{c} \begin{array}{ccccc} s_1 & s_2 & s_3 & s_4 & s_5 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0.4 & 0.3 & 0.2 & 0.1 \\ 0 & 1 & 0 & 0 & 0 \\ 0.1 & 0.9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

За допомогою вектору початкової вірогідності і матриці переходів можна вичислити стохастичний вектор $p\,(n)$ – вектор, складений з вірогідності $p\,(n)\,(i)$ того, що процес виявиться в змозі і у момент часу n. Отримати $p\,(n)$ можназа допомогою формули:

$$p\,(n) = p\,(0) \times P\,n$$

Вектори $p\,(n)$ при зростанні n в деяких випадках стабілізуються – сходяться до деякого імовірнісного вектору p, який можна назвати стаціонарним розподілом ланцюга. Стаціонарність проявляється в тому, що узявши $p\,(0) = \rho$, ми отримаємо $p\,(n) = \rho$ для будь-якого n [16].

Простий критерій, який гарантує збіжність до стаціонарного розподілу, виглядає таким чином: якщо усі елементи матриці перехідної вірогідності P додатні, то при n, що прагне до нескінченності, вектор $p\,(n)$ прагне до вектору p, тому, що є єдиним рішенням системи виду

$$p \times P = p.$$

Також можна показати, що якщо при якому-небудь позитивному значенні n усі елементи матриці $P\,n$ додатні, тоді вектор $p\,(n)$ все одно стабілізуватиметься [17].

1.3. Існуюче програмне забезпечення для визначення авторства тексту

Розглянемо приклади розробленого програмного забезпечення, яке виконує функцію визначення авторства тексту, та методи, на яких базується їх аналіз.

1.3.1. Система «Лінгвоаналізатор»

Д.В. Хмельовим [18, 19] було проведено ряд дослідів, за результатами яких було зроблено висновки щодо ефективності використання алгоритмів, який використовує ланцюги Маркова першого порядку та методів стиснення. З'ясовано, що вони дають гарні результати на довгих текстах і погані у порівнянні з іншими алгоритмами на текстах розмірами у 2 – 5 тисяч символів. Такий метод було реалізовано у системі «Лінгвоаналізатор».

Метод, запропонований Д.В. Хмельовим заснований на застосуванні відносної ентропії. Існує декілька способів розрахунку цієї величини, зокрема, шельфовий алгоритм, метод передбачення за частковим співпадінням та застосування індекса повторюваності.

Серед алгоритмів стиснення даних без втрат найбільш популярними є кодування Хафмана, арифметичне кодування, метод Бароуза-Вілера, та варіації методу Лемпеля-Зіва.

Необхідно зазначити, що серед алгоритмів, які спрямовані саме на стиснення текстів, найкраще себе показали метод передбачення за частковим співпадінням, який засновано на Макрівському ланцюгу та метод динамічного Марківського стиснення, який також засновано на ланцюгу Маркова.

Загалом, для експерименту, проведеному на системі «Лінгвоаналізатор» було відібрано 285 текстів 82 письменників. Кожен текст був попередньо оброблений, викинуто роздільники, та слова, що починаються з великої літери. Результати показали, що методи, засновані на ланцюгах Маркова,

визначають авторство тексту з ймовірністю 84% – 89%, що є прийнятним показником. Серед цікавих побічних результатів було те, що порівняння письменників різних історичних періодів давало погані результати, з чого можна зробити припущення, що історична епоха, коли було написано твір, суттєво впливає на точність результатів [18, 19].

1.3.2. Система «Атрибутор»

Як продовження підходу, який використовує у якості стилістичної характеристики бінарні буквосполучення, О.М. Тімашев запропонував використовувати тріади – трибуквенні буквосполучення. За такого підходу аналізу піддаються однобуквенні та двобуквенні службові слова, а це значна частина таких частин мови, як прийменник, частка, сполучник, які вважаються значущими стилеметричними характеристиками.

Саме через це двобуквенні сполучення, та сполучення з 4 та більше букв менш показові, що було виявлено в результаті досліду. На основі цього дослідження було побудовано програмний комплекс «Атрибутор», для автоматичного порівняння і класифікації текстів на основі триграм (трибуквенних сполучень).

База цієї системи містить твори 103 авторів та використовує експертну обробку текстів. У еталонну вибірку для системи було обрано романи, які отримані з електронних бібліотек. Вибірку було підібрано таким чином, щоб тексти різних письменників максимально відрізнялись одне від одного, а тексти одного письменника були максимально подібними.

Для обробки тексту цим програмним комплексом необхідно, щоб його об’єм складав не менше 6 сторінок. Це обмеження введене для того, щоб статистична вибірка була репрезентативною.

В основі алгоритму аналізу системи «Атрибутор» також лежать Марківські ланцюги. Точність цього аналізатора оцінюється у 80 – 85% [20].

1.3.3. Система «СМАЛТ»

Існують також підходи до аналізу текстів, які засновані на особливостях не лексичних, а синтаксичних, такі, як дерева залежностей та типів зв'язку. Результатом таких досліджень є реалізація системи атрибуції текстів «СМАЛТ» (Статичні методи аналізу літературного тексту), яка заснована на алгоритмах автоматизації морфологічного та статистичного аналізу [21, 22].

Система побудована з декількох блоків. Перший блок розбиває вхідний тест на лексичні одиниці, такі як розділ, абзац, речення, слово. Другий блок безпосередньо виконує аналіз перетвореного тексту та морфологічний розбір. Третій блок на основі морфологічного розбору виконує синтаксичний аналіз.

Базою для аналізу слугувала 81 стаття 19 століття з журналів «Время» та «Эпоха», авторство яких мали перевірити. Оскільки було достовірно відомо, що деякі зі статтею було написано Ф.М. Достоєвським, тож випробування проводились саме на авторство його статей.

В ході розробки була висунута гіпотеза про ефективність деяких методів аналізу текстів [23]. Зокрема, критерій Колмогорова-Смірнова ефективно себе показував при перевірці на співвідношення з заданим розподілом. Також ефективними виявились методи кластерного аналізу, де в якості основної характеристики текстів використовувалась матриця частот парного співпадіння граматичних класів слів.

Для проведення експерименту за допомогою критерію Стьюдента в якості параметрів було взято такі величини, як середня довжина слів у буквах, середня довжина речення у словах, індекс різноманітності лексики (відношення числа різних словоформ до числа словоуживанності). Проводились досліди з різними об'ємами вибірок (від 200 до 600 слів). В результаті було отримано значення критерію Стьюдента для всіх статей. Серед групи статей авторства Достоєвського було обрано максимальне

значення t - характеристики. Серед групи статей для перевірки було виключено тексти з характеристикою більше фіксованого.

При роботі з такими параметрами, як загальний розподіл довжини слова, загальний розподіл довжини речення, лексичний спектр тексту на рівні словника та лексичний спектр тексту на рівні тексту ставилось завдання визначення ймовірності того, що розподіл довжин слів в буквах у двох статтях, одна з яких – об'єднання статей Ф.М. Достоєвського, взято з однієї сукупності і можуть розглядатись як керовані одними й тими самими закономірностями.

Для цього було використано непараметричний критерій Колмогорова-Смірнова. Використовувались частотні словники на кожні 500 слів тексту. Всі словоформи розподілились за групами по частоті зустрічі у вибірці. Далі визначалось число словоформ в кожній групі, що означає розподіл частот на рівні тексту.

В результаті експериментів не вдалося виявити, чи є автором статей Достоєвський, так як обидві гіпотези (те, що автор Достоєвський, і те, що він не є автором) хибні.

У методі ієрархічної кластеризації використовувалися 2 міри відстані між об'єктами: Евклідова міра і міра Чебишева. Для визначення відстані між кластерами використовувалися методи ближнього і далекого сусіда. Дослідження проводилося на основі двох наборів ознак: основного, що складається з частин мови (16 ознак), і розширеного, з додаванням додаткових морфологічних параметрів, наприклад відмінок, рід і т.п. (156 ознак). Застосування методів кореляційних плеяд і ієрархічної кластеризації показало неефективність використання формально-граматичних параметрів для класифікації досліджуваних статей з метою вирішення завдання атрибуції, більш того, було доведено, що збільшення числа цих параметрів не покращує результати дослідження. Застосування методики «сильний граф»,

заснованої на вивченні закономірностей розташування частин мови в рамках пропозиції за певними параметрами, позбавила змоги чітко і однозначно відповісти на питання про приналежність ряду статей Ф.М. Достоєвському. Ще один недолік запропонованих методів полягає в тому, що завдання визначення авторства доводиться зводити до задачі побудови якісного і швидкого синтаксичного аналізатора. Останнє ж є досить важкою задачею навіть з сучасним розвитком технологій.

1.3.4. Система «Стилеаналізатор»

Проблему атрибуції текстів в роботах [24, 25] пропонується вирішувати за допомогою нейронних мереж і методів ієрархічної кластеризації. В якості міри порівняння матриць частот появи ознак в дослідженні використовувалася міра Кульбака і міра хі-квадрат. В роботі також показано, що міра Хмелева [18] є окремим випадком міри Кульбака. В [24] запропоновані підходи для порівняння стилів текстів за частотними ознаками з використанням гіпергеометричного критерію (двостороннього точного критерію Фішера) і критерію хі-квадрат. Під частотною ознакою мається на увазі будь-яка ознака стилю тексту, що допускає можливість знаходження частоти його появи в тексті (наприклад, число появи абзаців в тексті). На основі проведених досліджень розроблено програмний комплекс «Стілеаналізатор».

Проведено дослідження залежності якості класифікації текстів за авторством від обсягів текстових фрагментів за жанровими типами і джерелами за допомогою дерев рішень, методу Хмельова і методу з використанням нейронних мереж. В експериментах було взято два набори текстів: художніх творів (156 текстів, три підмножини: 30, 20 і 10 авторів) і газетних статтей (5 697 текстів, 57 журналістів за 2003 – 2004 рр.). Розглянуто

кількісні ознаки трьох рівнів: рівня букв, слів і речень. Всього 14 різних наборів ознак.

Було виявлено, що для різних текстів, з різним числом класів, для різних наборів ознак існує приблизно постійне мінімальне значення обсягу фрагментів за взятою класифікацією. Воно становить 30 000 – 40 000 символів, або 5 000 – 6 000 слів, або 400 – 600 речень.

Використовувалися нейронні мережі, які навчаються без вчителя і призначені для обробки великих масивів багатовимірної інформації – самоорганізаційні карти Кохонена (Self-organizing map – SOM). За останні роки цей напрямок є одним з найдинамічніших. За допомогою SOM-мереж вирішуються багато проблем класифікації, обробки природних мов, зображень, тестування і навчання. Незважаючи на широке використання, SOM-мереж не вистачає теоретичної обґрунтованості – вони спираються в основному на емпіричні результати.

В результаті було отримано висновок про те, що в разі вдалого знаходження універсального набору характеристик можна обробляти будь-яке число авторів і текстів (великі масиви інформації). Досить постійно модифікувати карту, додаючи нові твори, і оцінювати, як вони взаємодіють з вже обробленими.

Одним із серйозних недоліків методу є неможливість прогнозування успішного результату. Генетичний пошук на заданому наборі текстів може ніколи не знайти хороший варіант для поділу характеристик. Немає ніякого критерію того, чи в правильному напрямку рухається пошук, чи правильно він робить стрибки, чи накопичує корисну інформацію про простір, що досліджується. Дослідник сам повинен робити моніторинг пошуку і стежити за всіма кроками. Крім того, немає механізмів, що визначають, скільки часу залишилося до кінця роботи алгоритму, до того моменту, коли подальший пошук не принесе своїх результатів.

Іншою проблемою методу є його трудомісткість. Число завантажених текстів, яке безпосередньо впливає на якість пошуку, вимагає істотних ресурсів від обчислювальної системи (великий обсяг пам'яті і потужний процесор). Для знаходження по-справжньому універсальних характеристик необхідно обробити масивні корпуси текстів, щоб можна було з упевненістю заявити про їх універсальність.

Проведені експерименти показали, що метод Хмелева і його модифікації виграють як в швидкості навчання, так і в якості класифікації. Нейронні мережі дають порівняну якість, але сильно програють в швидкості. Дерева рішень забезпечують найгіршу якість класифікації, але при цьому дають наочний вид рішення і по ходу проводять відбір найбільш інформативних ознак.

1.3.5. Система «Авторовед»

Продовження досліджень щодо застосування нейронних мереж в поєднанні з методом опорних векторів при встановленні авторства текстів знайшло відображення в роботі [1]. Якщо задачу визначення авторства сформулювати як задачу класифікації, то одним з широко розповсюджених рішень є побудова бінарного класифікатора. Всі тексти, включно з навчальною частиною вибірки, розгортаються в дуже великий вектор, що індексується словами. Після цього є дві множини точок з навчальної вибірки в багатовимірному просторі: що належать даному автору і не належать автору. Для того щоб розділити ці множини, потрібно поділити простір на дві частини. Найпростіший спосіб зробити це – побудувати гіперплощину. Таку гіперплощину можна побудувати за допомогою методу опорних векторів (SVM – Support Vector Machines). Після цього для класифікації тексту з невідомим автором досить перевірити, в яку частину простору він потрапив.

Прикладами застосування методу опорних векторів при встановленні авторства є роботи [1, 26].

Методи класифікації за допомогою SVM значно перевершують численних конкурентів: кластерну класифікацію (коли документ співвідноситься до найближчої множини; в залежності від визначення відстані до множини – є багато варіантів реалізації цього методу – середньозважений, k найближчих сусідів і ін.), а також наївний Байєсовий класифікатор (який передбачає, що частоти слів у тексті є незалежними випадковими величинами).

В якості характерних ознак тексту для опису авторського стилю пропонується брати найбільш часті триграми символів і найбільш уживані(з найбільшою частотою) слова російської мови.

Як інструменти для атрибуції текстів в даній роботі були обрані штучні нейронні мережі архітектури багатошаровий перцептрон (MLP), мережі каскадної кореляції (CCN) і апарат машини опорних векторів (SVM). CCN дозволяють знизити часові витрати на навчання в порівнянні з перцептроном за рахунок алгоритму автоматичної побудови топології мережі. SVM є найбільш точним з існуючих методів класифікації і одночасно найшвидшим.

Підсумкове рішення про автора тексту приймається ансамблем класифікаторів за принципом мажоритарного голосування.

Основні результати проведених досліджень були отримані на корпусі, що складається з 215 прозових текстів 50 російських письменників. Тексти взяті з електронної бібліотеки М. Мошкова. Розмір кожного тексту становив понад 100 000 символів. Використовувалися вибірки об'ємом 10 00 – 100 000 символів (200 – 20 000 слів). Кількість навчальних прикладів кожного учасника бралося рівним 3, для тестування використовувалося по 1 вибірці автора. Експерименти для випадку 2, 5 і 10 авторів показали, що найбільш інформативними авторським ознаками є обмеження в 300 – 700 найбільш

частотних триграм і 500 найбільш частих слів. Автора можна визначити з точністю в середньому 0,95 – 0,98 при обсязі текстової вибірки 20 000 – 25 000 символів. При цьому починаючи з 10 000 символів машина опорних векторів показує кращі з трьох досліджуваних класифікаторів результати.

Встановлено, що використання при ідентифікації автора комбінації частот букв російської мови, знаків пунктуації, найбільш частих триграм символів і найбільш частих слів збільшує точність ідентифікації в середньому на 0,06 – 0,12 на обсягах тексту до 10 000 символів. У таблиці нижче наведено порівняння розглянутих програмних засобів (табл. 1).

Таблиця 1.1

Порівняння програмних засобів визначення авторства тексту

Назва	Метод	Зміна параметрів методу	Необхідний мінімальний об’єм тексту	Точність, %
Лінгвоаналізатор	Марківські ланцюги, відносна ентропія	Ні	40000 символів	84 – 89
Атрибутор	Маркавські ланцюги	Ні	20000 символів	80 – 85
СМАЛТ	Критерії Стьюдента, Колмогорова, кластерний аналіз	Ні	500 слів	невідомо
Стилеаналізатор	Марківські ланцюги, нейронні мережі	Так	30000 символів	90 – 98

Авторовед	Нейронні мережі, метод опорних векторів, QSUM	Так	20000 символів	95-98

1.4. Висновки

В даному розділі розглянуто теоретичні застави задачі визначення авторства тексту, існуючі методи рішення даної задачі, такі, як метод головних компонент, критерій Колмогорова-Смірнова, критерій Стьюдента, кластерний аналіз та метод Маркова. Також було розглянуто та порівняно програмні засоби, які реалізують методи вирішення задачі визначення авторства. Було виявлено, що формальні статистичні методи визначення авторства дають кращий результат за точністю та швидкодією. Також встановлено, що спільною рисою таких методів атрибуції є аналіз текстової інформації на основі частотних характеристик тексту, які мають вказувати на авторський інваріант.

2. МОДИФИКОВАНИЙ МЕТОД ВИЗНАЧЕННЯ АВТОРСТВА ТЕКСТУ НА ОСНОВІ ЛАНЦЮГА МАРКОВА

2.1. Способи попереднього оброблення текстових даних

Розглянемо декілька способів попереднього оброблення текстових даних для модифікованого методу визначення авторства тексту на основі методу ланцюгів Маркова. Залежно від режиму роботи та обраних характеристик основного методу попереднє опрацювання може модифікувати текст різним чином. Основне завдання такої обробки є видалення даних, які не впливають на роботу основного методу, або заважають йому.

2.1.1. Нормалізація тексту

Оскільки метод спрямований на аналіз лексичних одиниць, необхідно вилучити з тексту такі елементи, як розділові знаки. Також необхідно вилучити слова, що починаються з великої літери та пряму мову, оскільки такі структури, як власні імена та пряма мова може погіршити результат статистичного аналізу авторського стилю на основі використовуваних ним слів та буквосполучень. Отже, алгоритм попередньої обробки складається з таких етапів:

- видалення прямої мови;
- вилучення слів, що починаються з великих літер;
- вилучення розділових знаків.

Після такої обробки текст можна аналізувати на основі різних текстових характеристик, такі, як довжини слів, довжина речень, кількість різних граматичних конструкцій.

2.1.2. Вилучення службових частин мови

На наступному етапі, коли аналіз на основі слів проведено, необхідно прибрати частки, прийменники та сполучники. Оскільки вже було вирахувано

кількість та частоту використання таких частин мови, для подальшого аналізу вони не є необхідними.

2.1.3. Зведення форм слів

Наступний етап полягає у приведені слів до початкової форми. Оскільки у кожного слова є чимала кількість різних форм (відмінки, множина, звороти і т.д.), для статистичного аналізу ці характеристики не є суттєвою інформацією. Після цього етапу можна побудувати частотні характеристики тексту на основі вживання різних слів. При цьому слова «буква», «букви», «буквою» будуть вважатися одним і тим самим словом – «буква».

2.1.4. Вилучення пропусків

Наступні етапи аналізу мають на меті побудувати частотні характеристики на основі вживання *n*-грам буквосполучень та переходу з однієї букви до іншої, як випадок реалізації ланцюга Маркова. Отже, на виході ми отримаємо послідовність літер українського алфавіту, які можна аналізувати.

2.2. Опис та порівняння формальних статистичних методів для визначення авторства тексту

Як було показано вище, статистичні методи, засновані на частотних характеристиках тексту (зокрема, частоті біграм) не поступаються в точності визначення авторства методам, заснованих на нейронних мережах. До того ж, на відміну від останніх, вони не використовують синтаксичний аналіз, а отже є швидшими і легшими у реалізації.

Як було встановлено у першому розділі, всі статистичні методи використовують певні частотні характеристики тексту для побудови

авторського інваріанту. Розглянемо математичне підґрунтя методу ланцюгів Маркова та оцінки, які можна застосувати до частотних характеристик. Також проведемо порівняння ефективності використання різних частотних характеристик.

2.2.1. Метод ланцюгів Маркова

В останні десятиліття намітилася тенденція пошуку і виявлення характерних структур авторської мови шляхом застосування формально-кількісних, статистичних методів. Перші пробні кроки на цьому шляху зробив ще на початку XIX століття М.О. Морозов [27]. Цікаво, що тоді ж відомий математик А.А. Марков виступив з критикою М.О. Морозова [28]. А.А. Марков критикував М.О. Морозова за те, що він не провів ретельної статистичної перевірки тверджень щодо стійкості деяких елементів авторського стилю (наприклад, частки «не»). Прикладом правильного статистичного підходу А.А. Марков вважав своє дослідження в статті [29], де він вивчав розподіл частки голосних і приголосних серед перших 20000 букв «Євгенія Онєгіна». Відзначимо, що дана робота присвячена першому застосуванню «випробувань, пов'язаних в ланцюг», що одержали згодом назву ланцюгів А.А. Маркова. Вона (робота) надає історичну основу методу визначення авторства, викладену далі.

Позначимо через A деяку множину букв. Через A^k позначимо множину слів довжини k над алфавітом А. Нехай $A^* = I_k > -0A_k$. Позначимо довжину слова f O A^* через $|f|$.

Завдання визначення автора тексту можна сформулювати наступним чином [30].

Нехай задані n класів C_i, де $i = 0,\ldots,n-1$. У кожному класі C_i знаходяться послідовності $f_{i,j}$ O A^*, де $j = 1,\ldots,mi$, тобто

$C_i = \{f_{i,j} \mid j = 1, \text{j}, m_i\}$. Наше завдання полягає в тому, щоб віднести x 0 A^* до одного з класів C_i.

Припустимо, що послідовності букв $f_{i,j}$ є реалізаціями ланцюга Маркова з перехідною матрицею P^i. Побудуємо оцінку P^i. Позначимо через $h_{i,j,kl}$ кількість переходів букв $k->\ l$ в фрагменті $f_{i,j}$, припустимо $h_{i,kl}$ = $\text{e}_j h_{i,j,kl}$, а $h_{i,k}$= $\text{e}_l h_{i,kl}$. Припустимо $P_{kl}^i = {h_{i,kl}}/{h_{i,k}}$. Можливо, деякі P_{kl}^i дорівнюють нулю. Позначимо через Z_i множину таких упорядкованих пар (k, l), що $P_{kl}^i > 0$.

Припустимо, що x також є реалізацією ланцюга Маркова з матрицею перехідних ймовірностей P_q, де q невідомий параметр, який приймає одне із значень $1, \ldots, n$.

Позначимо через $V_{k,l}$ кількість переходів $k->\ l$ в x. Нехай також V_k = $e_l\ V_{k,l}$.

Позначимо через

$$L_i(x) = -e_{(k,l)}^{V_{k,l} * \ln(V_{k,l}/(P_{kl}^i * V_k))},$$

де сума береться по парам (k, l) 0 Z_i. Грубо кажучи, $L_i\ (x)$ дорівнює мінус логарифму імовірності x за умови, що x – реалізація ланцюга Маркова з матрицею перехідних ймовірностей P^i. Назвемо $t(x)$ оцінкою максимальної правдоподібності для невідомого параметра q:

$$t(x) = argmin_{i=0,\ldots,n-1} L_i(x). \qquad (2.1)$$

2.2.2. Метод Бернуллі

Схемою Бернуллі в теорії ймовірностей називається послідовність незалежних однаково розподілених випадкових величин. Формально ми можемо припустити, що послідовності $f_{i,j}$ і x є реалізаціями послідовності

незалежних однаково розподілених випадкових величин, що приймають значення в A, а x розподілений як величини класу η, де η – невідомий параметр.

Тоді оцінка (2.1) набуває вигляду

$$e(x) = argmin_i G_i(x),$$

де

$$G_i(x) = -e_k^{V_k \ln((V_k * h_i/(h_{i,k} * v))}, \quad (2.2)$$

де сума обчислюється за такими k, що $V_k > 0$, а $V = e_k v_k$, $h_i = e_k\ h_{i,k}$, грубо кажучи, виконуючи оцінку $\eta(x)$ ми виконуємо частотний аналіз тексту. Статистичний експеримент, наведений у статті [11] показує, що оцінка $e(x)$ суттєво програє у точності оцінці $t(x)$.

Отже, проведемо порівняння ефективності визначення авторства тексту на основі різних частотних характеристик.

Вихідний корпус текстів в результаті попередньої обробки при визначенні авторства тексту представлений у варіантах (а) – (г).

а) аналіз пар букв в їх природних послідовності в тексті – в словах (в тій формі, в якій вони вжиті в тексті) і прогалини між ними;

б) аналіз пар букв в послідовності букв в приведених (словникових, лематизованих або початкових) формах слів; наприклад, попереднє речення в такому випадку постає у вигляді «пара буква в послідовність буква приведений словниковий лематизований або початковий форма слово»;

в) аналіз пар найбільш узагальнених («неповних») граматичних класів слів, частин мови (окрім службових), в їх послідовності в реченнях тексту – іменники, дієслова, прикметники і т. п.: всього 9 традиційно виділених граматичних класів слів – частин мови та 3 інших умовних категорій на кшталт «кінець речення», «скорочення», «неясний клас»; категорія «неясний клас» введена в зв'язку з тим, що розбір по граматичним класам проводився

автоматично (і покривав більше 99% всіх оброблених слів), але деякі слова (наприклад, з друкарськими помилками) не піддавалися автоматичній обробці, а тому їх граматичний клас залишався неясним.

г) аналіз пар менш узагальнених («повних») граматичних класів слів (а саме таких семантико-граматичних розрядів, як одухотворені іменники, неживі іменники, прикметники якісні, відносні, присвійні і т. п.).

Була проведена перехресна перевірка методу на матеріалі 385 текстів 82 авторів.

Результати перехресної перевірки вказаними методами представлено в табл. 2.1 – 2.2.

Таблиця 2.1

Результати перехресної перевірки правильності визначення автора тексту з використанням послідовності букв

Випадок (а)			Випадок (б)		
Словоформи тексту			Приведені форми		
R	Пари букв	Одиночні	*R*	Пари букв	Одиночні
0	282/385	27/385	0	240/385	12/385
1	21/385	55/385	1	29/385	40/385
2	9/385	25/385	2	17/385	19/385
3	5/385	24/385	3	9/385	16/385
4	5/385	17/385	4	6/385	16/385

≥ 5	63/385	237/385	≥5	84/385	282/385
M	3,38	12,69	*M*	4,77	17,88

Продовження табл. 2.1

Таблиця 2.2

Результати перехресної перевірки правильності визначення автора тексту з використанням послідовності граматичних класів

Випадок (в)			Випадок (г)		
Узагальнені граматичні класи			«Повні» граматичні класи		
R	Парні	Одиночні	*R*	Парні	Одиночні
0	235/385	128/385	0	15/385	6/385
1	31/385	43/385	1	21/385	12/385
2	16/385	29/385	2	9/385	6/385
3	8/385	15/385	3	12/385	6/385
4	11/385	17/385	4	19/385	8/385

≥5	84/385	153/385	≥5	309/385	347/385
M	5,43	10,13	*M*	17,76	31,93

Відмінність варіанту (а) від початкового тексту (в цьому одна з відмінностей даного дослідження від існуючих) полягає в тому, що відкинуті всі слова, для яких не вдалося автоматично визначити граматичний клас (а отже, і знайти словникову форму). Це було зроблено, щоб можна було порівнювати результати з результатами варіанту (б). При цьому весь текст перетворений в послідовність слів і пропусків між ними, вся пунктуація була відкинута, і, крім того, були викинуті всі слова з великої літери (включаючи слова, з яких починаються речення; такий прийом дозволяє значно збільшити точність визначення авторства; це поліпшення пов'язане з тим, що відкидаються імена літературних героїв, які, як правило, не співвідносяться зі стилем автора літературного тексту). У використовуваному алфавіті буква «г» склеюється з буквою «ґ», в результаті чого разом з пропуском вийшло 33 літери.

Кожна буква кодувалася своїм номером: буква «а» відповідала 1, «я» відповідала 32. Пропуску зіставлявся код 0. Загальна кількість буквоутворень склало 96209964. Загальна кількість різних пар букв, виявлених на уже згадуваному масиві текстів, склало 1011 (з потенційно можливих 33 х 33 = тисячу вісімдесят дев'ять).

Очевидно, 1011 трохи перевершує кількість різних пар букв, які дійсно зустрічаються в текстах українською мовою. Це є результатом певної кількості помилок в електронних версіях книг і може призводити до погрішностей в обчисленнях. Щоб отримати деяку оцінку цього шуму, були відібрані всі буквосполучення, які навряд чи зустрічаються в українській

мові, і їх виявилося 121. Загальна кількість вживань цих буквосполучень 38495, що становить близько 0,04% від загального числа вживань буквосполучень, чим цілком можна знехтувати.

Перетворення початкових текстів для отримання їх у вигляді варіантів (б), (в) і (г) здійснено на основі автоматичного класифікатора, розробленого О.В. Кукушкіною [31], в лабораторії загальної та комп'ютерної лексикології і лексикографії на основі граматичного словника Залізняка і академічних граматик.

Варіант (в) базується на інформації, одержуваної при використанні самого загального граматичного класу словоформи, тобто інформації про частини мови (частки, прийменники, вигуки, з'єднувальні союзи, інші союзи, дієслова, займенники, прислівники, прикметники, іменники, числівники, предикативи, компаративи, модальні прислівники).

Варіант (г) базується на інформації, одержаній при обліку лексико-граматичного розряду слів даної частини мови (одухотворені іменники, неживі іменники і т. п.).

Наведемо деяку статистику по текстам, які відповідають варіантам (б) – (г). У варіанті (б) загальна кількість букв склала 110704464. Кількість різних пар букв в початкових формах слів склала 1029 (з потенційно можливих 33 х 33 = 1089). Як видно, воно збільшилося в порівнянні з варіантом (а). Такий ефект пов'язаний з переведенням непрямих форм в початкові.

У варіантах (в) і (г) кількість елементів склала 20262449. При цьому у варіанті (в) кількість різних пар елементів – 302 (з потенційно можливих 18 х 18 = 324). У варіанті (г) кількість різних пар елементів склала 8124 (з потенційно можливих 112 х 112 = 12544).

Основним результатом проведеного дослідження є те, що використання граматичної інформації в рішенні задачі визначення дійсного автора тексту є не тільки осмисленим, але і досить ефективним, а в деяких випадках

порівнянним з використанням інформації про пари букв що зустрічаються в тексті, як це було показано раніше.

2.3. Модифікований метод визначення авторства тексту на основі ланцюга Маркова

Для більш ефективного визначення авторства тексту пропонується модифікований метод оснований на методі визначення авторства, запропонованого А.А. Марковим та методі визначення інваріантів автора (групи авторів). Оскільки метод Маркова був розглянутий нами в п. 2.2.1, зупинимось на методі визначення інваріантів автора чи групи авторів.

Для віднесення вхідного тексту до одного з досліджуваних класів необхідно сформувати відповідні інваріанти.

Під авторським інваріантом ми розуміємо кількісну характеристику літературних текстів (деякий параметр), який:

а) однозначно характеризує своєю поведінкою твори одного автора або невеликої кількості «близьких авторів»

б) приймає істотно різні значення для творів різних груп авторів.

Бажано, щоб кількість «різних груп» було досить великою, і щоб кожна група об'єднувала відносно мало схожих, близьких за стилем авторів.

Проте, розмаїття граматичних структур, що беруть участь у формуванні літературних текстів, сильно ускладнює пошуки таких інваріантів. Вже прості обчислювальні експерименти показують, що виявлення числових характеристик, що розрізняються різних авторів – складне завдання. Справа в тому, що коли людина пише книгу, то істотну роль відіграють не тільки підсвідомі, а й свідомі чинники. Наприклад, частота вживання автором рідкісних та іноземних слів, може, звичайно, служити певним показником його стилю, ерудиції. Однак цей показник легко контролюється автором на свідомому рівні, оскільки рідкісні і іноземні слова вставляються в текст

нечасто і кожен раз автор спеціально наголошує про себе: «тут я вставляю іноземне або рідкісне слово». В результаті, як незаперечно свідчать конкретні підрахунки, використовувати цю числову характеристику в якості авторського інваріанта не можна. Вона контролюється автором, сильно змінюється і письменник може легко міняти її від твору до твору.

Звідси видно, що кількісна оцінка індивідуальних відмітних особливостей автора – вельми нетривіальне завдання.

Сформулюємо точніше, якими властивостями повинен володіти авторський інваріант.

Шукана числова характеристика повинна відповідати наступним природним вимогам.

1) Вона повинна бути досить «масовою», інтегральною, щоб слабко контролюватися автором на свідомому рівні.

Іншими словами, вона повинна бути його «несвідомим параметром», що корениться настільки глибоко, що автор навіть не замислюється про нього. А якби навіть замислився, то не зміг би довго його контролювати і в результаті досить швидко повернувся б у попередній сталий і типовий для нього стан.

2) Шуканий параметр повинен зберігати «постійне значення» для творів даного автора. Тобто, мати невелике відхилення від середнього значення (слабо коливатися) протягом усіх його книг. Саме ця властивість і дозволяє говорити, що даний параметр є інваріант.

3) Нарешті, параметр повинен впевнено розрізняти між собою різні групи письменників. Іншими словами, має існувати достатня кількість авторських груп, що помітно відрізняються один від одного значеннями інваріанта.

Таким чином, класифікація тексту, тобто визначення його відповідності деякому способу створення, автору (групі авторів) або іншому класу, може

бути представлена в формальному вигляді, заснованому на теоретико-множинному підході.

T – множина всіх текстів.

K – множина класів текстів, $|K| = n$.

$A = \{a_1, a_2, \ldots, a_n\}$ – множина інваріантів класів текстів в рамках розв'язуваної задачі, $|A| = n$.

T_a – множина текстів, яким підтверджено деякий інваріант, тобто тексти відомого походження або авторства.

T' – множина текстів, яким не підтверджено інваріант, тобто сукупність текстів невідомого класу.

При цьому виконується: $T = T_a \cup T'$; $T_a \cap T' = \emptyset$.

X – кінцева множина досліджуваних текстових характеристик, множина визначається наступним чином:

$X = \{x \mid x - \text{досліджувана текстова характеристика}\}$; $|X| = m$.

Інваріант $a_j \in A$ являє собою масив упорядкованих пар виду: <текстова характеристика $x_i \in X$; значення текстової характеристики x_i для даного інваріанту z_{ij}>. Тобто $a_j = (< x_1, z_{1j} >, \ldots, < x_i, z_{ij} >, \ldots, < x_m, z_{mj} >)$, де $i = 1, \ldots m$; $j = 1, \ldots n$; z_{ij} – деяке числове значення, формат і діапазон якого було встановлено відповідно до текстової характеристики, в окремих випадках в якості z_{ij} можуть виступати діапазони значення характеристики: $z_{ij} = [z_{ij}\ min;\ z_{ij}\ max]$.

Тоді на декартовому добутку множини текстів T і множини інваріантів А може бути задано бінарне відношення $R \subset T \times A$ таке, що виконується tR_a, якщо деякий текст $t \in T$ відповідає інваріанту $a \in A$, тобто текст t відноситься до класу, якому відповідає інваріант a, або текст t написаний автором, якому відповідає інваріант a.

З огляду на всі попередні позначення, можна записати умову того, що деякого вхідного тексту $t \in T$ відповідає інваріант $a \in A$.

Відношення tR_a виконується, якщо значення текстових характеристик досліджуваного тексту t відповідають або наближені в певній мірі до значень характеристик $x_i \in X$ інваріанта a. При цьому ступінь наближеності значень встановлюється автором методу в кожному конкретному випадку і повинна бути обґрунтована експериментальними даними.

Для виявлення «несвідомого параметра» – авторського інваріанта, слабко або взагалі не контрольованого письменниками, вивчено наступні кількісні характеристики текстів.

1) Довжина речення, тобто середня кількість слів у реченні, підрахована для кожної вибірки.

2) Довжина слів, тобто середня кількість складів у слові, підрахована для кожної вибірки.

3) Загальна частота вживання службових слів – прийменники, сполучники, часток, тобто процентний вміст службових слів в кожній вибірці.

4) Частота вживання іменників, тобто їх процентний вміст в кожній вибірці.

5) Частота вживання дієслів, тобто їх процентний вміст в кожній вибірці.

6) Частота вживання прикметників (у відсотках).

7) Частота вживання прийменника «в» (у відсотках).

8) Частота вживання частки «не» (у відсотках).

9) Кількість службових слів у реченні, тобто середня кількість займенників, прийменників і часток у реченні.

10) Частота вживання біграм.

11) Частота вживання триграм.

Деякі з перерахованих параметрів розглядалися і раніше. Однак запропонований параметр – частота всіх службових слів – є, безперечно, новим.

В якості критерію стабілізації узято наступний принцип. Обсяг вибірки (кількість символів для аналізу береться кроками по 2000 символів) збільшувався до тих пір, поки не виявлявся параметр, для якого середня величина його відхилень від середніх значень уздовж творів усіх досліджуваних письменників виявлялася істотно менше амплітуди коливань параметра між текстами різних авторів.

Іншими словами, для кожного учасника обчислювалося відхилення параметра від середнього значення, а потім ці відхилення осереднювалися по всім авторам. Розшукувався параметр, для якого це останнє число істотно менше різниці між максимальним і мінімальним значеннями параметра по всіх досліджуваних письменниках.

В результаті дослідження встановлено, що параметри з 1 по 8, окрім 5 параметру при збільшенні обсягу вибірки не стабілізуються. Отже, у якості характеристик, на яких буде формуватися інваріант було обрано частоту вживання службових частин мови, частоту вживання слів, а також частотний розподіл біграм і триграм.

Запропонований метод визначення авторства на основі ланцюгів Маркова складається з наступних етапів:

- попереднє оброблення тексту;
- аналіз отриманого результату методом ланцюгів Маркова;
- оцінка ступеня схожості заданого тексту з авторськими інваріантами;
- за умови однозначної відповіді (ймовірність співпадіння з одним із інваріантів вища 95%, співпадіння з іншими менша 50%) перерахунок відповідного інваріанту.

2.4. Висновки

Таким чином, в даному розділі нами розглянуто методи попередньої обробки текстових даних, та їх відмінності. Було розглянуто та порівняно формальні статистичні методи визначення авторства тексту. Виявлено, що розгляд тексту як послідовності випадкових біграм та триграм дає суттєво кращий результат, аніж аналіз на основі граматичних класів. Також порівняно оцінку Бернулі з оцінкою методу Маркова, остання з яких виявилась кращою. Було розглянуто декілька характеристик, за якими можна формувати авторський інваріант та запропоновано модифікований метод визначення авторства тексту на основі ланцюгів Маркова.

3. ОПИС РОЗРОБЛЕННОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Основні вимоги до програмного забезпечення

Основними вимогами до застосунку, що розроблюється, з точки зору користувача та розробника є:

1. Функціональна повнота. Передбачає реалізацію в повному обсязі поставлених замовником завдань зі зберігання, введення, редагування і надання інформації. Створена система повинна в повній мірі виконувати завдання по підтримці прийняття рішення, надавати всі необхідні функції роботи користувача, зберігати потрібну інформацію, і виключати зайву.

2. Зручність. Створений застосунок повинен бути зручним, мати зрозумілий, зручний, «дружній» інтерфейс. Здійснювати максимально можливий набір дій при мінімальному втручанні користувача.

3. Надійність. Створений програмний продукт повинен мати високу відмовостійкість, бути стабільним в роботі і надавати достовірну інформацію.

4. Технічна ефективність. Досягається шляхом правильного проектування реляційної бази даних, що дозволяє зберігати великі обсяги інформації на малоємностних носіях.

5. Адаптивність. Дозволяє з мінімальними витратами проводити модифікацію продукту, в залежності від додаткових потреб замовника.

6. Вартість. Розроблювана система повинна мати невисоку вартість і доступність.

Розглянемо більш детально кожну позицію.

Оцінка програмного забезпечення, з точки зору сторони замовника і виконавця може здійснюватися з використанням різних показників якості. Для досягнення поставленої мети розробки виберемо наступні, на наш погляд основні, критерії:

Функціональна повнота.

Даний показник характеризує ступінь задоволення потреб користувача в сенсі можливості вирішення конкретних завдань, що стоять перед ним, тобто це набір атрибутів, що визначає призначення, основні необхідні і достатні функції програмного забезпечення, зазначені в технічному завданні замовника або потенційного користувача.

Деталізується на такі характеристики:

– придатність для застосування (відповідність призначенню);

– точність;

– здатність взаємодіяти з середовищем;

– відповідність нормам;

– безпека (резервне копіювання даних, захист від можливості доступу до даних третіх осіб і ін.).

Оптимальним значенням функціональної повноти є одиниця. Якщо цей показник більше 1, то є функціональна надлишковість; в іншому випадку виникає функціональна недостатність. У будь-яких випадках, коли функціональна повнота не дорівнює 1, програмний виріб придатний до експлуатації.

Надійність.

Один з найбільш неоднозначних, суперечливих і важливих критеріїв якості програмного забезпечення, під яким розуміють перш за все забезпечення досить низької ймовірності появи проблем, які призводять до неможливості використовування застосунку. Саме в силу ненадійності програмних продуктів виникають величезні витрати на супровід ПЗ, пов'язаних, перш за все з експлуатацією програмних систем.

Можна виділити два основних аспекти надійності:

– відсутність в готовій програмі помилок проектування і програмування;

– захищеність програми від непередбачених умов експлуатації.

Надійна програма не реагує на натискання недозволених клавіш; ненадійна – може завершитися аварійно, поставивши користувача в скрутне становище. Навіть не здійснюючи неправильних дій, користувач, тим не менш, може ввести неприпустиме поєднання вихідних даних, що призводять до аварійного завершення програми. Надійна програма повинна виконувати відповідні перевірки і не допускати обчислень з цими даними, попереджаючи користувача про неможливість виконання його вимог і повертаючись в попередній стан.

Зручність.

З точки зору задоволення попиту споживача цей показник якості може іноді мати вирішальний вплив на вибір того чи іншого програмного продукту. Цей критерій показує, наскільки зручно користувач-непрофесіонал може застосовувати його в своїй повсякденній роботі. Складність такої оцінки полягає у відсутності конкретних числових показників, якими можна охарактеризувати зручність а також необхідність зворотного зв’язку від користувачів. Проте, зручність використання програмного продукту можна охарактеризувати наступними показниками:

– зручність для користувача інтерфейсу, зручність розташування та подання часто використовуваних елементів екрану, способів введення даних і ін.;

– простота освоєння, трудові і тимчасові витрати на освоєння засобів;

– адаптованість до конкретних вимог користувача;

– кількість інформації, що пред'являється системою, яку необхідно переробити користувачеві;

– кількість дій, використаних користувачем при роботі з системою;

– простота використання.

Технічна ефективність.

Цей показник характеризує обсяг необхідних ресурсів обчислювальної системи для безперебійного функціонування застосунку. Зокрема, це обсяг оперативної пам'яті, обсяг зовнішньої пам'яті, час роботи процесора, які залучаються компоненти операційної системи. Практично всі ці показники можуть отримати числову оцінку, і природно, можна сказати, що з двох програм з однаковими функціональними властивостями краще та, яка споживає менше ресурсів. Таким чином, ефективність ПЗ слід розглядати за такими двома параметрами:

– час відгуку та швидкодія;

– споживання ресурсів, вимоги до мінімального розміру зовнішньої і оперативної пам'яті, продуктивності процесора (тактова частота та кількість операція за одиницю часу), що забезпечують прийнятний рівень продуктивності.

Адаптивність.

Мається на увазі можливість модифікації ПЗ в разі зміни параметрів завдань, що вирішуються, операційного оточення, апаратурного складу або взагалі типу ЕОМ. При зміні характеру і параметрів вирішуваних завдань, як правило, потрібно змінити кількісні і структурні параметри програми, тобто змінити її вихідний текст. Якщо ж програма не споряджена вихідними текстами або потужним засобом щодо внесення змін до параметрів програми без порушення вихідних текстів, переносимість (адаптованість), близька до нуля. Цей параметр найбільше залежить від ефективного проектування архітектури системи та процесів розробки ПЗ. Даний показник якості застосунків характеризують такі приватні показники:

– кроссплатформенність;

– сумісність з сучасними версіями найрозповсюдженіших операційних систем;

– структурованість;

– замінність;

– легкість інсталяції;

– відповідність нормам по переносимості та інсталяції;

– впровадженність.

Вартість.

З точки зору споживача слід розрізняти два аспекти показника: вартість придбання і вартість експлуатації. В цьому плані користувач завжди стоїть перед вибором: придбати типовий готовий програмний продукт або замовити індивідуальну розробку. У першому випадку вартість придбання порівняно невелика, проте адаптація ПЗ, його налаштування на клас вирішуваних завдань потребуватиме певних витрат. Створений за індивідуальним замовленням програмний продукт, як правило, помітно дорожче готового, але повніше відповідає вимогам користувача. В цьому випадку вартість його придбання значно вище, але вартість налаштування і експлуатації порівняно нижче, ніж у першому варіанті. До того ж розроблений на замовлення продукт працює оптимальніше за рахунок спрямування саме на вирішення задач замовника, без вирішення додаткових задач.

3.2. Опис обраних засобів розробки програмного забезпечення

3.2.1. Засоби проектування програмного продукту

Дуже важливим етапом розробки програмного забезпечення є саме проектування. Від якості виконання цього етапу залежить подальша ефективність розробки та результати проекту в цілому. Звісно, що у сучасних

реаліях важко спроектувати систему, яка б не зазнавала змін безпосередньо під час її створення. Але однією з задач проектування є саме внесення адаптивності в систему, для порівняно швидкої зміни певних функцій, без втручання у сторонні функції, які мають залишитись без змін. Для створення таких проектних рішень існує досить багато потужних систем проектування програмного забезпечення, які широко розповсюджені серед проектних команд розробників.

Сукупність методів та підходів до проектування програмного забезпечення , які покликані забезпечити вищезазначені критерії якості розробленого продукту називають CASE-методами (англ. computer-aided software engineering). Відповідно, програмні системи для автоматизації цього процесу називають CASE-засобами.

Одним з найвідоміших сучасних CASE-засобів є Rational Rose від компанії Rational Software Corporation. Власне, саме ця компанія була одним із ініціаторів розробки і стандартизації мови UML (Unified Modeling Language) у 1997 році. Цей проект підтримали, і після релізу версії 1.1 мова стала стандартом для проектування інформаційних систем у таких компаніях, як IBM, Microsoft, та інших.

Rational Rose є одним з лідерів ринку за рахунок широких можливостей, які він надає користувачам. Цей програмний продукт можна застосовувати впродовж усього процесу розробки програмного продукту, оскільки його функціональні можливості дозволяють спростити та автоматизувати майже усі етапи побудови інформаційної системи будь-якої складності.

Основними можливостями Rational Rose є:

- проектування систем практично будь-якої складності;
- підтримка усіх стандартних діаграм та нотацій мови UML, які існують у версії 2.0;

- автоматизація контролю розробки, порівняння відповідності моделей;
- автоматична генерація документації для проекту, представлена власним інструментом SoDA (Rational Software Documentation Automation);
- автоматична генерація коду високорівневими мовами програмування;

Розглянемо кожну можливість більш детально.

Засоби проектування.

Rational Rose спрямована саме на моделювання систем. Головною перевагою процесу проектування є те, що діаграми, сценарії, моделі, які описуються на цьому етапі є досить зрозумілими як і замовнику, так і розробникам програмного забезпечення. Звісно, є більш спеціалізовані типи діаграм, такі, як діаграми класів, схема бази даних, але більшість діаграм покликана прибрати непорозуміння між замовником та виконавцями.

При цьому, система дозволяє розбивати діаграми високого рівня на діаграми більш низького, що дозволяє бачити як систему цілком, так і переглядати окремі аспекти та модулі. До складу Rational Rose входить досить зручний графічний інтерфейс, який дозволяє зручно проводити проектування підсистем та контролювати цей процес.

Такий підхід є великою перевагою, оскільки дозволяє на ранніх етапах помітити такі недоліки систем, як висока зв’язність модулів, що унеможливлює адаптивність системи. Також це дозволяє провести аналіз «вузьких місць» системи в цілому та окремих модулів, зокрема.

Підтримка UML.

Оскільки UML фактично є стандартом для систем проектування, підтримка цієї мови у CASE-системах є дуже важливою складовою.

Переважна кількість типів діаграм, яка є у мові версії 2.0 підтримується у сучасних версіях Rational Rose.

Хоча, мова UML має деякі недоліки. Основною проблемою є неточність її семантики та певна надмірність. Але система проектування дещо полегшує використання стандарту мови, оскільки не підтримує надлишкові діаграми, які є рідковживаними та має модуль перевірки побудованих діаграм, що дозволяє накласти обмеження на семантику UML.

Автоматизація контролю розробки.

Оскільки проектування великої системи є досить складним та довготривалим процесом, у ході розробки можуть виникати різні непередбачувані ситуації, такі як:

- зміна функціональних вимог;
- необхідність відновити попередню версію застосунку;
- зміна архітектури модулів.

Для спрощення контролю над такими ситуаціями система проектування інтегрується з системою конфігураційного керування PVCS.

Ця система дозволяє розбивати моделі певного рівня на підмоделі. Для кожної підмоделі можна виконувати такі операції:

- завантаження підмоделі у оперативну пам’ять;
- вивантаження підмоделі з оперативної пам’яті;
- збереження підмоделі в постійній пам’яті, як окремий файл;
- встановлення захисту від модифікації;
- зміна підмоделі на нову;
- версіонування моделей.

Оскільки кожна модель може мати свої підмоделі, зміна однієї з них може спровокувати зміну моделі в цілому. Тож захист від модифікації

підмоделей дозволяє легко убезпечити програміста від випадкової модифікації модулів системи, у яких зміни не були передбачені.

Версіонування ж дозволяє легко повернутися до попередньої версії системи за необхідності. Також це дозволяє легко вносити зміни в архітектуру згідно з новими функціональними вимогами.

Автоматична генерація документації.

Однією з досить великих проблем розробки інформаційних систем є документування. Оскільки документація системи після її розробки практично неможлива внаслідок великої складності системи та постійних її змін, документування зазвичай ведеться прямо в процесі розробки. Це вимагає досить багато часу.

Система Rational Rose дозволяє згенерувати документацію на етапі проектування системи автоматично за допомогою системи SoDA. Вона має такі можливості:

– Автоматичне вилучення інформації з файлів, створених різними інструментальними засобами. SoDA аналізує структуру інформації, що зберігається тими системами, з якими вона інтегрована, а сама інформація доступна їй через API цих систем.

– Збереження при «перекомпіляції» тексту і графіки, введених користувачем вручну в текстовому процесорі. Якщо користувач, скажімо, в Microsoft Word, додав якісь коментарі або ілюстрації в згенерований за допомогою SoDA документ, то при перебудуванні даного документа SoDA його не зіпсує.

– Налаштування шаблонів, за якими генерується документація. За допомогою зручного візуального редактора можна створювати шаблони, відповідні всіляким зовнішнім стандартам (таким як ISO 9000, IEEE, MIL-STD-498 і DOD-STD-2167A) або внутрішнім стандартам компанії.

– Синхронізація з джерелами і перевірка актуальності документації. Зв'язки між окремими частинами документації та вихідними файлами зберігаються. Тому, по-перше, SoDA може відстежувати зміни, що відбуваються з джерелами, на основі яких була в останній раз сформована документація, а по-друге, користувач може з будь-якої секції документа швидко отримати доступ до джерел, інформація з яких використовується в цій секції.

– Часткова «перекомпіляція» великих документів. Проектна документація до масштабних програмних систем може досягати гігантських обсягів. Тому в SoDA передбачена можливість «перекомпілювати» тільки такі частини документації, які дійсно втратили актуальність.

– Збір інформації з численних і різнорідних джерел.

– Документування всіх етапів роботи над проектом.

– Перевірка дотримання вимог, що пред'являються до системи, що розроблюється. SoDA дозволяє сформувати таблиці, з яких можна зрозуміти, наскільки отримані результати відповідають вимогам, визначеним на початкових етапах проектування.

– Підтримка формування шаблонів і звітів.

Генерація коду.

Дуже суттєвою перевагою системи Rational Rose є можливість генерувати код на основі сформованої моделі системи. Це дозволяє не витрачати багато часу на програмування усіх елементів системи, натомість, лише виправляючи та поліпшуючи автоматично згенерований код.

Такі системи досить сильно впливають як на кошторис проекту (зменшуючи витрати на програмістів), так і на час виконання проекту.

Система генерації коду, що надається Rational Rose, генерує код високого рівня. У базовому пакеті є декілька мов, найуживанішою з яких є мова C++.

Водночас, ця можливість має ряд недоліків. По-перше, це не замінює роботу програміста, який може написати набагато оптимальніший код. Також існує обмеження на програмні засоби, які можуть бути використані у ході такої генерації. Оскільки система може мати різні вимоги, існує ймовірність того, що деякі елементи системи буде важко інтегрувати між собою. Тож таку систему треба використовувати обдумано.

3.2.2. Обрані засоби розробки

Для розробки програмного забезпечення, що реалізує модифікований метод визначення авторства на основі ланцюгів Маркова, було обрано засоби платформи .NET, а саме:

– .NET Standart 2.1;

– Мова C# 7.2;

– .NET Framework 4.7.2.

У якості середовища розробки було обрано Microsoft Visual Studio 2017 Community Edition.

У якості сервера баз даних було обрано MSSQL Server 2012.

У якості системи керування базами даних було обрано Microsoft Management Studio 2017;

Розглянемо детальніше обрані засоби, та їх застосування у процесі розробки.

Платформа .NET, .NET Standart та .NET Framework.

.NET – програмна платформа, яка була розроблена компанією Microsoft у 2002 році. Основними компонентами платформи є CLR (Common Language Runtime) та FCL (Framework Class Library).

CLR – це віртуальна машина, на якій виконується будь-яка програма, написана на платформі .NET. Основною перевагою платформи є саме підхід до виконання програм. Програма, яка написана мовою, що підтримується платформою (специфікація мов описана в CLI) компілюється в проміжний байт-код, який називається CIL (Common Intermediate Language). Після цього CLR перетворює проміжний код у машинний код конкретного процесору JIT-компілятором (Just-In-Time) і виконує його. Завдяки такому підходу досягається ряд цілей:

– можливість написання однієї системи різними мовами (головна умова – відповідність CLI);

– компіляція програми оптимальним чином для цільового пристрою (наприклад, при компіляції програми на телефоні, де обмежено запас пам’яті, програму буде оптимізовано так, щоб вона витрачала менше пам’яті);

– підтримка кроссплатформенності, за умови реалізації CLR для інших платформ;

– відсутність необхідності перекомпіляції програми при її перенесенні на інший пристрій, оскільки переноситься саме проміжний код, який після того буде оптимізовано.

FCL – бібліотека класів, яка включає в себе всі класи, які можуть бути використані усіма мовами, які підтримує сама платформа. Такі базові типи знаходяться у ядрі FCL, що має назву BCL (Base Class Library). Окрім цього, FCL містить у собі класи, які використовуються у таких технологіях .NET, як ADO.NET, Windows Forms, ASP.NET, WCF та інші.

Впродовж багатьох років основним напрямом розвитку платформи був .NET Framework. Це програмна платформа, яка була розрахована під використання виключно на базі операційних систем сімейства Microsoft Windows.

Наразі платформа продовжує розвиватись у вигляді модульної платформи .NET Core, яка була розрахована на кроссплатформенність зі стадії її проектування. Поточна версія цієї платформи 2.2, і вона підтримує веб-інтерфейс та консольний інтерфейс. У версію 3.0, вихід якої заплановано у 2019 році, буде включено підтримку технологій Windows Forms, WPF, які входять до компоненту Windows Desktop, що буде частиною .NET Core SDK. Також варто зазначити, що .NET Core є open-source проектом, код якого у відкритому доступі знаходиться на GitHub.

.NET Standard являє собою офіційну специфікацію інтерфейсів API .NET, які повинні бути доступні у всіх реалізаціях .NET. .NET Standard створена для того, щоб підвищити узгодженість екосистеми .NET.

.NET Standard надає наступні важливі можливості:

– Визначає уніфікований набір API-інтерфейсів BCL для реалізації всіма реалізаціями .NET незалежно від робочого навантаження.

– Дозволяє розробникам створювати переносимі бібліотеки, які можуть використовуватися в різних реалізаціях .NET, за допомогою одного набору API-інтерфейсів.

– Дозволяє скоротити або навіть усунути умовну компіляцію загального джерела через API-інтерфейсів .NET (тільки для API операційної системи).

Різні реалізації .NET реалізують конкретні версії .NET Standard. Кожна версія реалізації .NET орієнтована на використання максимальної підтримуваної нею версії .NET Standard. Це також означає, що вона

підтримує і попередні версії. Наприклад, платформа .NET Framework 4.6 реалізує .NET Standard 1.3, тобто надає всі API-інтерфейси, визначені в стандартах .NET Standard версій з 1.0 до 1.3. Аналогічним чином платформа .NET Framework 4.6.1 реалізує .NET Standard 1.4, а .NET Core 1.0 - .NET Standard 1.6.

Тож зараз екосистема платформи .NET має такий вигляд (рис. 3.1).

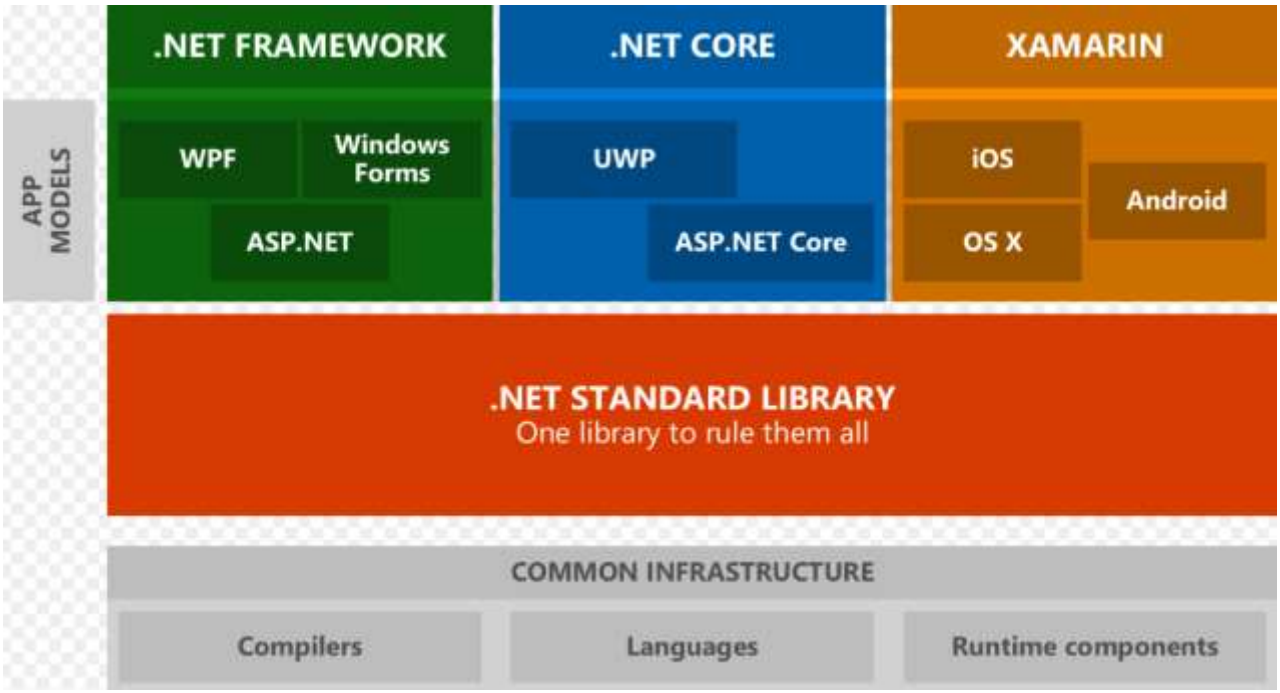


Рис. 3.1. Екосистема платформи .NET

Мова програмування C#.

C# – об’єктно-орієнтована мова програмування високого рівня. Була розроблена і випущена компанією Microsoft разом з програмною платформою .NET. Має C-подібний синтаксис. Розроблялась на основі таких мов, як Java, Delphi, C++ та інших. Розробники мови намагались проаналізувати проекти, розроблені на основі попередників і, користуючись цим досвідом, взяти найкращі риси з цих мов, і відкинути ті, які зарекомендували себе, як проблемні.

Мова має строгу статичну типізацію, що дозволяє зменшити кількість помилок при написанні програм. Завдяки цьому також було створено типізовані посилання на функції – делегати. На основі делегатів можна створювати систему подій, яка набула широкого користування у клієнтських інтерфейсах.

Усі типи даних мови спадкуються від класу System.Object. На відміну від Java, де примітивні типи даних виділені окремо, і для них є високорівневі «обгортки» у вигляді класів, примітиви у мові C# - це просто псевдоніми класів з базової бібліотеки класів. Різниця між примітивними типами та класами у тому, що примітиви зберігаються за значенням у стеку, у той час, як класи зберігаються у керованій купі, а у стеку – лише посилання на конкретні об’єкти.

Мова C# має керовану пам’ять, очищенням якої опікується Garbage collector – збирач сміття. Програміст не має змоги напряму очищати пам’ять, має лише змогу трохи коригувати роботу збирача сміття. Для некерованих ресурсів, таких, як потоки читання, запису файлів, мережеві сокети і т.д. передбачено інтерфейс IDisposable.

Оскільки мова об’єктно-орієнтована, вона підтримує всі принципи об’єктно-орієнтованого програмування, такі, як абстрагування, інкапсуляція, спадкування, поліморфізм, тощо. На відміну від мови C++, кожен клас може мати тільки одного предка. Проте, для забезпечення підтримки багатьох можливостей поведінки (певна заміна спадкуванню від багатьох предків), кожен клас може реалізовувати будь-яку кількість інтерфейсів. Інтерфейс – спеціальний об’єкт, який може мати у своєму складі методи, ітератори та індексатори, але не може мати полів. При цьому неможливо створити об’єкт інтерфейсу, але можна перевірити, чи є певний об’єкт «типом інтерфейсу», тобто, чи він реалізує його.

Для доступу до інкапсульованих полів мова C#, на відміну від мов Java та C++ має спеціальний об’єкт – властивість. Властивість дозволяє об’єднати у собі методи get та set для доступу та зміни даних поля відповідно. Також перевагою є те, що властивість може мати логіку, проте для користувача робота з нею виглядає, як робота зі звичайним полем.

Окрім цього, мова має спеціальний синтаксис для побудови запитів до колекцій – LINQ. За допомогою цим запитам можна робити вибірки з колекцій даних, їх трансформацію та інші операції, схожі на SQL-запити.

Для доступу до бази даних використовується ORM система Entity Framework Core. Він має широкі можливості керування базами даних з коду програми. Підтримує парадигми Code-fist, Database-first. Для доступу та вибірки даних використовується синтаксис LINQ, Entity Framework при цьому сам будує SQL-запити.

Мова має багато так званого синтаксичного «цукору» – конструкцій, які спрощують написання коду. Також саме середовище Visual Studio має вбудовану систему IntelliSense, яка дозволяє суттєво пришвидшити написання за рахунок пропозицій автозаповнення. Також у середовищі розробки є сніппети – комбінації клавіш, які дозволяють швидко написати якусь широковживану конструкцію, наприклад «ctor» – стандартний конструктор без параметрів, «prop» – автовластивість, тощо.

Окрім викладених вище властивостей, мова C# володіє також багатьма іншими, які спрямовані на спрощення роботи програміста та на поліпшення якості та читабельності коду. Окрім цього мова активно розвивається, що привносить у синтаксис додаткові зручності. Завдяки цьому, а також платформі .NET та .NET Core, зокрема, ці технології широко використовуються у написанні як невеликих застосунків, так і великих корпоративних систем. Не останню роль у цьому відіграє широке

розповсюдження хмарних систем, оскільки .NET Core має вбудовані можливості інтеграції з хмарними технологіями Microsoft Azure.

3.3. Опис розробленого програмного забезпечення

В рамках даної роботи розроблене наступне програмне забезпечення:

– система, що реалізує модифікований метод визначення авторства текстів, що базується на ланцюгу Маркова;

– застосунок із графічним інтерфейсом для визначення авторства текстів.

3.3.1. Система, що реалізує модифікований метод визначення авторства текстів, що базується на ланцюгу Маркова

Функціональні можливості розробленого програмного комплексу реалізовані у вигляді трьох взаємопов'язаних підсистем, кожна з яких складається з сукупності модулів. На рис. 3.2 приведена структура системи. Нижче подано короткий опис модулів, які представлено на рис. 3.2.

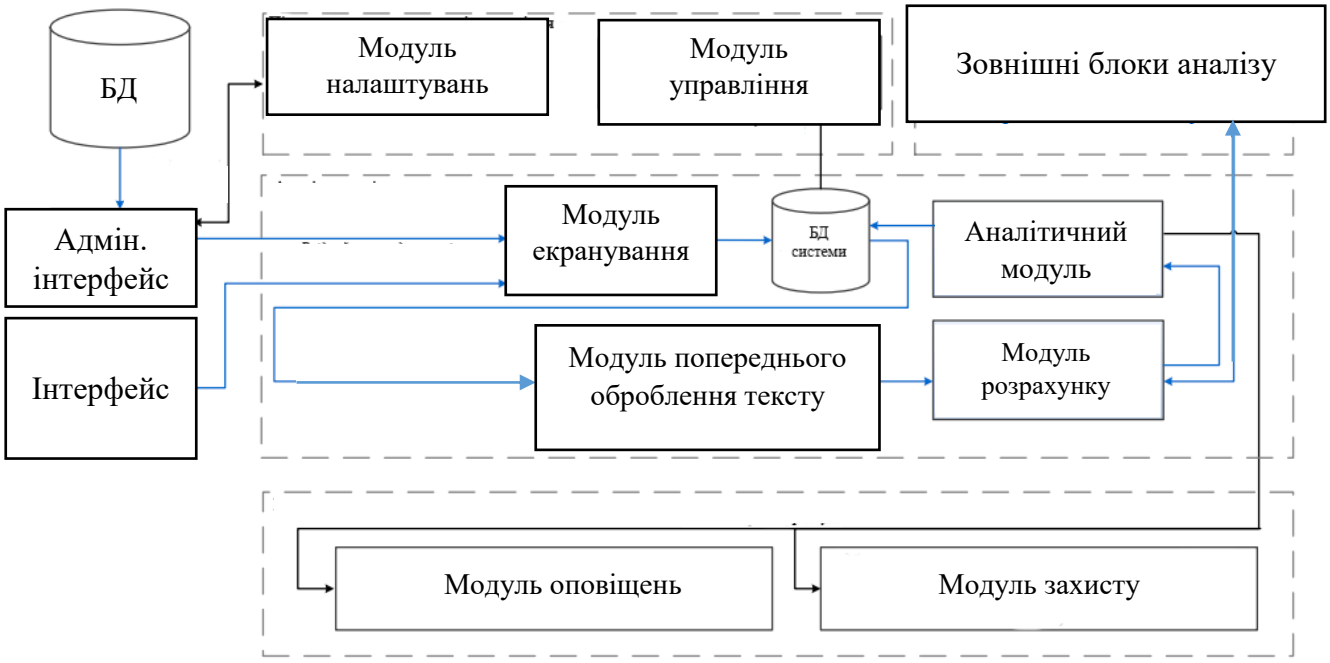


Рис. 3.2. Архітектура програмного засобу, що дозволяє визначати авторство вхідного тексту

Модуль екранування.

Даний модуль відповідає за екранування всіх потенційно небезпечних символів і команд, не дозволяючи поміщати в SQL-запит керуючі структури і ідентифікатори, введені користувачем.

Модуль попереднього оброблення тексту.

Наступним етапом є попереднє оброблення тексту, що представляє собою автоматизований процес підготовки вмісту до подальшого аналізу шляхом його зведення до формату вхідних даних запропонованого методу. Способи попереднього оброблення тексту наведено у п. 2.1 даної роботи. Відповідно, модуль реалізує описані вище методи

Модуль оповіщень.

Модуль оповіщення відповідає за обробку виключних ситуацій та логування подій, надаючи повну інформацію про всі дії, що відбуваються у системі.

Модуль розрахунків характеристик.

Розрахунок чисельних значень характеристик вхідного тексту – найважливіший етап роботи програмного засобу, так як його підсумком є набір значень, який буде використаний для прийняття рішення про походження тексту. Згідно позначенням запропонованого методу модуль формує набір значень A'. На вхід модуля подається досліджуваний текст, представлений у вигляді масиву слів, а також текст у вигляді рядка. На виході – значення характеристик даного тексту.

Аналітичний модуль.

Аналітичний модуль отримує дані з модулю розрахунків та порівнює з авторськими інваріантами, що зберігаються у базі даних. При цьому

результати записуються у локальну базу даних, дані з якої потрапляють до основної в кінці роботи методу.

Зовнішні блоки аналізу.

Додаткові блоки, які підключаються до системи для виконання інших видів аналізу. Використовуються для порівняння швидкодії та точності методів.

3.3.2. Застосунок із графічним інтерфейсом для визначення авторства текстів

В якості кінцевого застосунку для автоматичного визначення авторства текстів розроблено застосунок з графічним інтерфейсом. Такий варіант реалізації користувацького інтерфейсу дозволяє легко керувати введенням початкових даних та власне процесом прийняття рішення.

Структура інтерфейсу застосунку складається з однієї форми, на якій розміщені основні блоки введення даних, їх відображення та блоку навігації у вигляді головного меню (рис. 3.3).

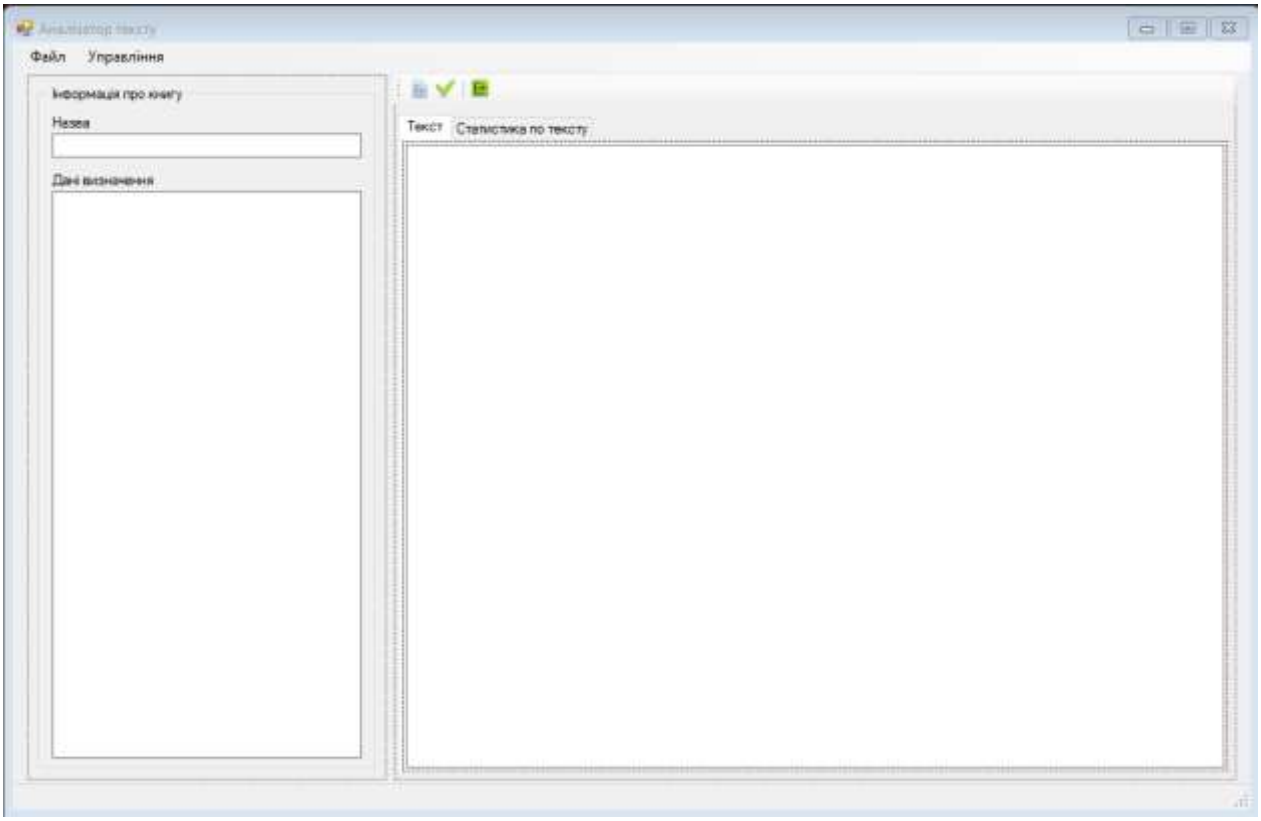


Рис. 3.3. Форма застосунку

Для вибору тексту для розгляду використовується пункт меню «Файл», де обирається відповідний текстовий файл, після чого текст відображається у головному вікні, а також виводиться ймовірність належності тексту певному автору (рис. 3.4).

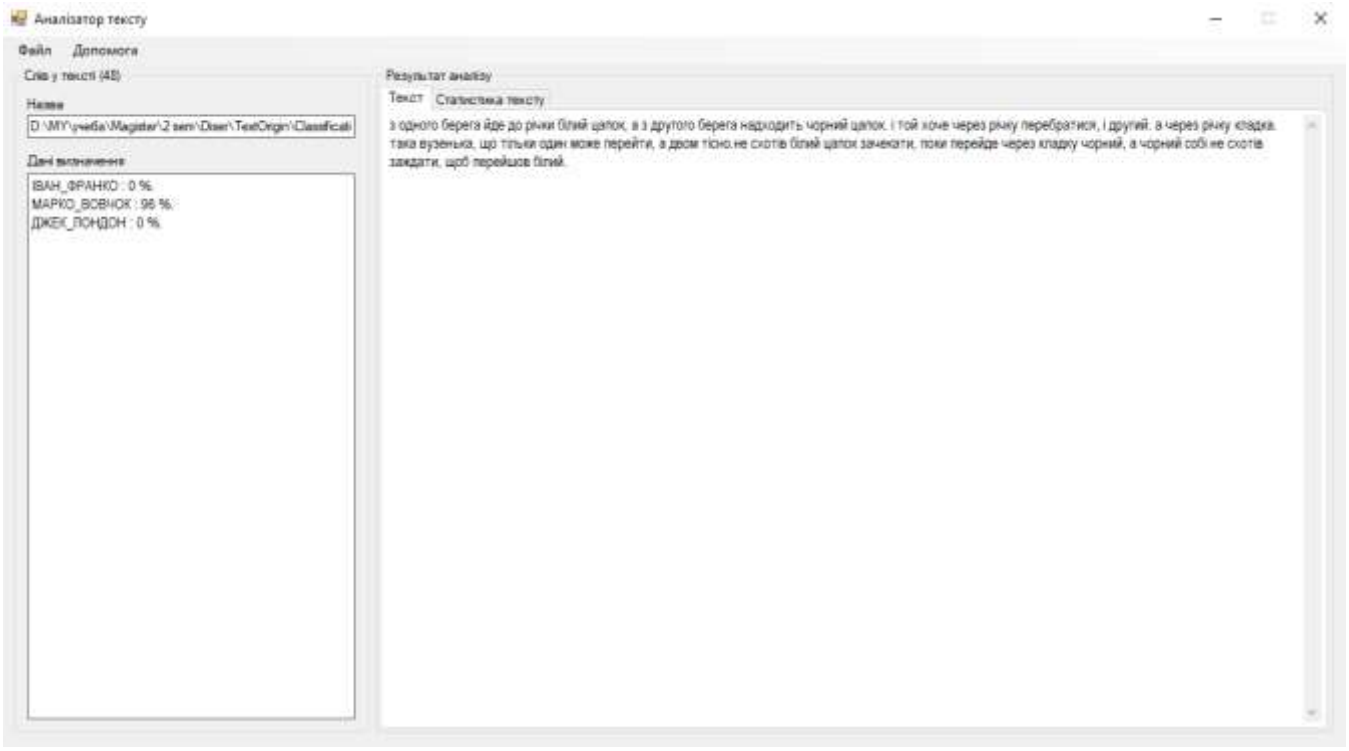


Рис. 3.4. Форма з завантаженим текстовим файлом

Також можна перейти на вкладку статистика тексту, і побачити частотний розподіл характеристик, які притаманні конкретному автору. Приклад зображено на рис. 3.5.

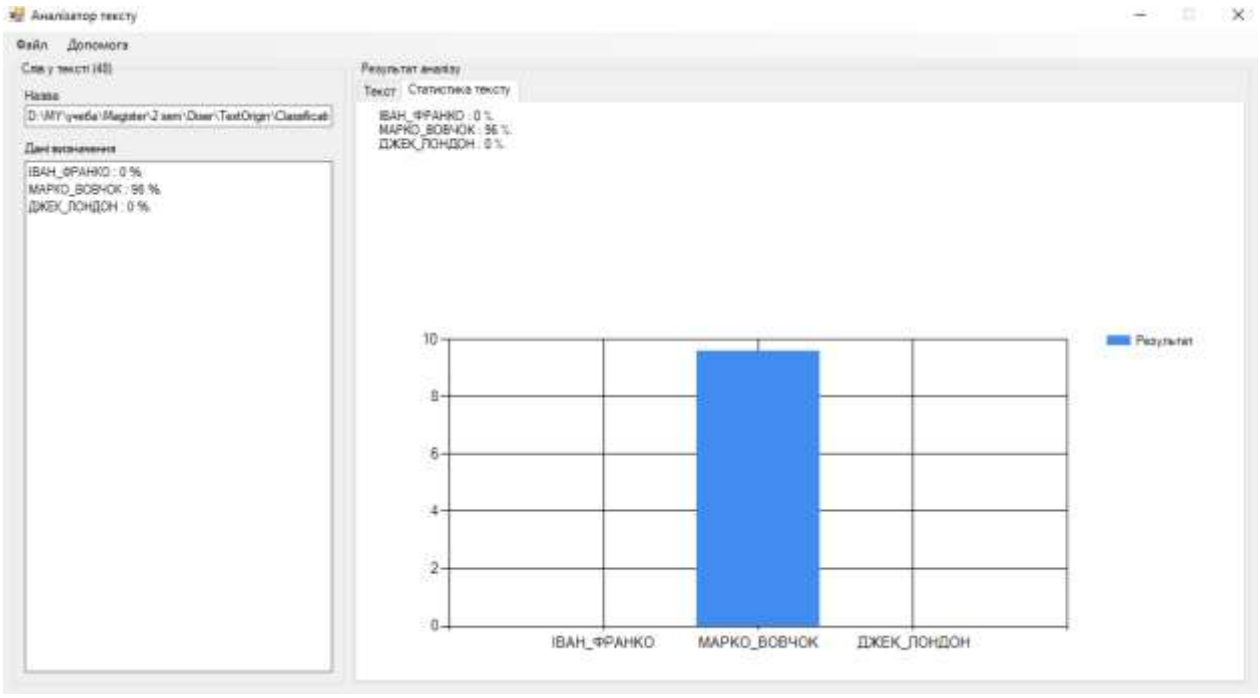


Рис. 3.5. Розподіл частотних характеристик, притаманних кожному автору

Для наочності було обрано фрагмент тексту маленької довжини, так, щоб він підпадав тільки під конкретного автора.

Застосунок написаний на об'єктно-орієнтованій мові програмування C #, що працює в середовищі виконання .NET Framework 4.7.2. Він реалізований у вигляді модулів, що дозволяє додавати нові функції до застосунку, не порушуючи його структуру.

Усі модулі для реалізації самого методу було написано на основі стандарту .NET Standart 2.1 у вигляді бібліотек класів. Це дозволяє використовувати і змінювати класи незалежно від інтерфейсу.

3.4. Висновки

В рамках даної роботи розроблено програмне забезпечення, яке призначене для визначення авторства текстів та реалізує модифікований метод ланцюга Маркова.

Можливо зробити наступні висновки:

1. Розроблене програмне забезпечення можливо використовувати для автоматизованого визначення авторства текстів.
2. Розроблений застосунок надає можливості роботи з розробленими класами, що реалізовують бізнес-логіку.
3. Архітектура розробленого програмного забезпечення дозволяє легко додавати нові модулі, що реалізують попереднє оброблення текстових документів та реалізацію модифікованого методу визначення авторства текстів.
4. Використані програмні засоби для розробки дозволяють використовувати розроблені модулі аналізу тексту на будь-якій платформі, на якій встановлено програмне забезпечення, яке підтримує .NET Standart 2.1.
5. Завдяки використанню засобів платформи .NET, для використання бібліотек можна використовувати будь-який користувацький інтерфейс, як консольний, так і графічний.

4. АНАЛІЗ ЕФЕКТИВНОСТІ МОДИФІКОВАНОГО МЕТОДУ ПРИЙНЯТТЯ РІШЕНЬ

4.1. Критерії оцінки ефективності

Загалом, основними критеріями при оцінці методів автоматизованого визначення авторства текстів є критерії точності, часу та пам'яті. Згідно з завданням, нас найбільше цікавить показник точності, оскільки модифікований метод розроблено задля підвищення точності визначення авторства тексту.

Наприклад, у дослідженні Д.В. Хмельова використовувався показник рангу оцінки подібності. Він міг мати значення від 0 до $n-1$, де n – кількість авторів. Якщо показник 0 – значить, метод вказав автора тексту як найбільш подібного за стилем. Якщо показник 1 – значить дійсний автор на другому місці за подібністю.

В модифікованому методі автоматизованого визначення авторства текстів, що базується на ланцюгу Маркова, результуючим показником є відсоток подібності стилю тексту, що аналізується, до стилю всіх авторів, які є у базі. Отже, найбільш ймовірним автором є той, для кого цей відсоток є найбільшим.

Метою дослідження було підвищення точності методу автоматизованого визначення авторства текстів, тому критерій точності буде основним, за яким буде проводитись оцінка та порівняння методів. Вторинним критерієм буде критерій часу, бо завжди треба співвідносити збільшення точності та потенційне зменшення швидкодії.

4.2. Дані, що використовуються для оцінки ефективності

В силу того, що в основі методу лежить аналіз та класифікація текстової інформації, усі текстові дані надаються українською мовою. Використання української мови дозволяє використовувати до неї усі методи попередньої обробки та усі підходи, описані в розділі 2.

Також для оцінки мінімальної довжини тексту, яка необхідна для релевантного аналізу, текст розбивається на групи по 2000 символів в кожній. Обсяг вибірки збільшувався до обсягу повного тексту.

До початку тестування необхідно було наповнити базу авторами та для кожного оючислити авторський інваріант. Для цього обчислення використовувався модифікований метод визначення авторства та розроблений програмний засіб.

Було взято 48 авторів. Для кожного автора було взято 10 текстів різного обсягу. Для кожнонго тексту було обчислено всі обрані характеристики та вирахуване середнє арифметичне за кожною характеристикою. Для спрощення аналізу характеристики вважаються рівнозначними, отже, 100% співпадіння однієї характеристики дає 25% схожості тексту з авторським інваріантом.

Уривок з тексту, що використовується для аналізу:

«І Дрогобича неминула холера. Особливо Лан утерпів від неї більше, як другі передмістя, чи то тому, що тут затхлий та нечистий воздух помагав ширенню зарази, чи, може, тому, що люди, стіснені густо в одних хатах, легко одні від других заражувалися. Жиди, жидівки, а найбільше малі діти падали, мов трава під косою, умирали серед глуші, тихо, потайно, по кутах та закамарках. Кілько їх там перемерло, то лиш бог один знає. Котрі були маєтніші, повиїздили при наближенні зарази в гори, на здоровіший воздух, – але зараза і там їх ді гнала, може, лиш де сотий успів вернути назад. Але у

Германової матері грошей не було, зарібку не стало і хліба, нічого. Вона серед загальної тривоги валандалася попід хату, без пам'яті від переляку і голоду, раз в раз заводила дивними голосами, доки й сама не впала на землю, заражена. Герман тямить добре, як прибіг до неї і з дитинячою цікавістю зблизився к тому тілу, посинілому, покорченому, близькому лютої смерті. Йому ще і нині живо стоїть перед очима вираз її лиця, такий безграничноболісний, перекривлений та змінений, що аж йому, малому, мороз перейшов по тілі. Він тямить кождий її рух, кожде її слово в тоті страшні хвилі вічної розлуки. Насамперед вона кивнула рукою, щоби не приступав близько, – материнська любов, хоть під грубою оболічкою, не загибла в ній, а проявилася в хвилю найтяжчої муки. Її рука, простягнена, безвладно упала на землю, а Герман бачив, як всі жили, всі сустави стягались, то випручувалисудорожно, як вона дрижала з холоду, а попід шкіру щораз видніше набігала синя, ба зеленкувата муравиця.

– Герш, – прохрипіла вона, – не підходи... до... мене!..

Хлопець стояв мов отуманілий. В тій хвилі він дуже мало і дуже невиразно розумів, що воно таке діється. Шарпане судорогами тіло матері почало перекачуватися на всі боки.

– Герш!.. чесно жий! – простогнала нещаслива, ледве дишучи. В тій хвилі упала лицем на землю.

Герман стояв, боячися приступити до неї, а не менше боячися утікати.

– Води! води! – прохрипіла конаюча, але Герман не міг рушитися з місця, його пам'ять щезла на тот час. Як довго він так стояв, два кроки віддалений від матері, того й сам не знає. Навіть не може собі нагадати, хто і як пробудив його з того остовпіння, коли і куди спрятали трупа: все то

пожерло вічне забуття, вічна непам'ять.» (Проза, Іван Франко «Boa constrictor»).

4.3. Приклади попередньої обробки текстових даних

Застосуємо розглянуті у розділі 2 методи попередньої обробки текстових даних.

4.3.1. Нормалізація тексту

Після виконання операції нормалізації тексту, які включають в себе такі кроки, як вилучення знаків пунктуації, прямої мови та слів з великої літери, текст, який оброблюється набув наступного вигляду:

«неминула холера утерпів від неї більше як другі передмістя чи то тому що тут затхлий та нечистий воздух помагав ширенню зарази чи може тому що люди стіснені густо в одних хатах легко одні від других заражувалися жидівки а найбільше малі діти падали мов трава під косою умирали серед глуші тихо потайно по кутах та закамарках їх там перемерло то лиш бог один знає були маєтніші повиїздили при наближенні зарази в гори на здоровіший воздух але зараза і там їх ді гнала може лиш де сотий успів вернути назад Але у матері грошей не було зарібку не стало і хліба нічого серед загальної тривоги валандалася попід хату без пам'яті від переляку і голоду раз в раз заводила дивними голосами доки й сама не впала на землю заражена тямить добре як прибіг до неї і з дитинячою цікавістю зблизився к тому тілу посинілому покорченому близькому лютої смерті ще і нині живо стоїть перед очима вираз її лиця такий безграничноболісний перекривлений та змінений що аж йому малому мороз перейшов по тілі тямить кождий її рух кожде її слово в тоті страшні хвилі вічної розлуки вона кивнула рукою щоби не приступав близько материнська любов хоть під грубою оболічкою не загибла в ній а проявилася в хвилю найтяжчої муки рука простягнена безвладно упала

на землю а бачив як всі жили всі сустави стягались то випручувалисудорожно як вона дрижала з холоду а попід шкіру щораз видніше набігала синя ба зеленкувата муравиця

стояв мов отуманілий тій хвилі він дуже мало і дуже невиразно розумів що воно таке діється судорогами тіло матері почало перекачуватися на всі боки

стояв боячися приступити до неї а не менше боячися утікати

довго він так стояв, два кроки віддалений від матері того й сам не знає не може собі нагадати хто і як пробудив його з того остовпіння коли і куди спрятали трупа все то пожерло вічне забуття вічна непамять».

4.3.2. Вилучення службових частин мови

Наступний етап – вилучення всіх допоміжних частин мови: сполучників, часток, прийменників.

«неминула холера утерпів неї більше як другі передмістя тут затхлий нечистий воздух помагав ширенню зарази може люди стіснені густо в одних хатах легко одні других заражувалися жидівки найбільше малі діти падали трава косою умирали серед глуші тихо потайно кутах закамарках їх перемерло бог один знає були маєтніші повиїздили при наближенні зарази гори здоровіший воздух зараза там їх дігнала може сотий успів вернути назад матері грошей було зарібку стало хліба нічого серед загальної тривоги валандалася попід хату пам'яті переляку голоду раз раз заводила дивними голосами сама не впала землю заражена тямить добре прибіг неї дитинячою цікавістю зблизився тому тілу посинілому покорченому близькому лютої смерті нині живо стоїть перед очима вираз її лиця такий безграничноболісний перекривлений змінений йому малому мороз перейшов тілі тямить кождий її рух кожде її слово страшні хвилі вічної розлуки вона кивнула рукою

приступав близько материнська любов хоть грубою оболічкою загибла ній проявилася хвилю найтяжчої муки рука простягнена безвладно упала землю бачив як всі жили всі сустави стягались то випручувалисудорожно як вона дрижала холоду попід шкіру щораз видніше набігала синя ба зеленкувата муравиця

стояв мов отуманілий тій хвилі він дуже мало дуже невиразно розумів воно таке діється судорогами тіло матері почало перекачуватися всі боки

стояв боячися приступити неї менше боячися утікати

довго він стояв, два кроки віддалений матері того сам знає може собі нагадати хто пробудив його того остовпіння коли куди спрятали трупа все пожерло вічне забуття вічна непам'ять».

4.3.3. Зведення форм слів

На цьому етапі всі зміни слів, як то відмінки або множина, вилучаються, всі слова змінюються до інфінітиву. Після цього даний текст набуває такого вигляду:

«неминула холера утерпіти неї більше як другі передмістя тут затхлий нечистий воздух помагає ширенню зараза може люди стіснення густо в одних хата легко одна другий заразитися жидівка найбільше мала дитина падати трава коса умирати серед глуш тихо потайно кут закамарка їх помирати бог один знає були маєтніший повиїздити при наближення зараза гора здоровіший воздух зараза там їх доганяти може сотий успів вернути назад матір гроші було зарібок стало хліб нічого серед загальна тривога валандатися попід хата пам'ять переляк голод раз раз заводити дивний голос сама не впала земля заражена тямить добре прибігти неї дитиняча цікавість зблизитися тому тіло посиніле покорчене близький люта смерть нині жива стояти перед очі вираз її лице таке безграничноболісне перекривлене змінене

йому малий мороз перейти тіло тямить кождий її рух кожде її слово страшне хвиля вічна розлука вона кивнути рука приступити близько материнська любов хоть груба оболічка загибла ній проявитися хвиля найтяжча мука рука простягнена безвладно упала земля бачити як всі жити всі сустав стягатись то випручувати судорожно як вона дрижати холод попід шкіра щораз видніше набігати синя ба зеленкувата муравиця

стояв мов отуманіле тій хвиля він дуже мало дуже невиразно розуміти воно таке діється судорога тіло матір почало перекачуватися всі бік

стояв боятися приступити неї менше боятися утікати

довго він стояти, два крок віддалений матір того сам знати може собі нагадати хто пробудити його того остовпіння коли куди спрятати трупа все пожерле вічне забуття вічне непам'ять».

Оскільки текст написано досить давно і досить специфічною мовою, алгоритм може неправильно визначати всі змінені форми а також виправляти їх неправильно.

4.3.4. Вилучення пропусків

На цьому етапі необхідно вилучити всі пропуски у тексті, щоб вийшов просто один рядок, що складається з букв українського алфавіту. При цьому прибираються також символи табуляції та символи кінця рядку. Необхідно зазначити, що цей етап обробки іде не після всіх попередніх, а виконується після першого етапу, щоб зберегти порядок та поєднання букв, як їх писав автор.

«неминулахолераутерпіввіднеїбільшеякдругіпередмістячитотомущотуз атхлийтанечистийвоздухпомагавширеннюзаразичиможетомущолюдистіснені густоводниххатахлегкооднівіддругихзаражувалисяжидівкианайбільшемалідіт

ипадалимовтравапідкосоюумиралисередглушітихопотайнопокутахтазакамарк
ахїхтамперемерлотолишбогодинзнаєбулимаєтнішіповиїздилипринаближенніз
аразивгориназдоровішийвоздухалезаразаітамїхдігналаможелишдесотийуспівв
ернутиназадАлеуматерігрошейнебулозарібкунесталоіхлібанічогосередзагальн
оїтривогиваландаласяпопідхатубезпам'ятівідперелякуіголодуразвраззаводила
дивнимиголосамидокийсаманевпаланаземлюзараженатямитьдобреякприбігдо
неїіздитинячоюцікавістюзблизивсяктомутілупосиніломупокорченомублизько
мулютоїсмертіщеінініживостоїтьпередочимавиразїїлицятакийбезграничнобо
лісний перекривленийтазміненийщоажйомумаломуморозперейшовпотілітями
тькождийїїрухкождеїїсловототістрашніхвилівічноїрозлукивонакивнуларуко
ющобинеприступавблизькоматеринськалюбовхотьпідгрубоюоболічкоюнезаги
блавнійапроявиласявхвилюнайтяжчоїмукирукапростягненабезвладноупалана
землюабачивяквсіжилив сісуставистягалисьтовипручувалисудорожнояквонад
рижалазхолодуапопідшкірущоразвиднішенабігаласинябазеленкуватамуравиц
ястоявмвотуманілийтійхвилівіндужемалоідуженевиразнорозумівщовонотакед
ієтьсясудорогамитіломатеріпочалоперекачуватисянавсібокистоявбоячисяприс
тупитидонеїанеменшебоячисяутікатидовговінтакстоявдвакрокивіддаленийвід
матерітогойсамнезнаєнеможесобінагадатихтоіякпробудивйогозтогоостовпінн
яколиікудиспряталитрупавсетопожерловічнезабуттявічнанепамять».

Є недолік у такому методі аналізу, оскільки вилучення відбувалося після вилучення прямої мови та всіх слів, що починаються з великої літери, які можуть бути просто початком речення, а не власною назвою. Але оскільки аналіз іде на великих за обсягами текстами, невелика різниця між аналізом початкового тексту без змін та аналізом результату цього етапу обробки буде згладжуватись.

4.4. Результати аналізу ефективності розробленого методу

При аналізі ефективності модифікованого методу автоматизованого визначення авторства текстів, що базується на ланцюгу Маркова, оскільки, як описано у розділі 2, він дає найкращі результати за критерієм точності та швидкості для даної задачі.

Для реалізації базового методу автоматизованого визначення авторства текстів, що базується на ланцюгу Маркова використовувалось розроблене та описане у розділі 3 програмне забезпечення.

Для моніторингу швидкодії використовувались стандартні компоненти платформи .NET – stopwatchs, оскільки цей критерій не був обраний як основний. При більш серйозному акценті на цих критеріях, можливо, доречним є використання бібліотеки BenchmarkDotNet, що є дуже потужним інструментарієм для аналізу швидкодії програмних засобів.

Спочатку було виконане порівняння оригінального та модифікованого методу на одному тексті, задля вивлення , яка кількість символів потрібна кожному методу для релевантого аналізу. Відсоток подібності вказаний для дійсного автора (колонки 2 та 3). Результати цього порівняння представлені у табл. 4.1.

Таблиця 4.1

Результати проведення тестування

Довжина тексту та відсоток схожості	ланцюг Маркова	Модифікований ланцюг Маркова	Час (базовий метод)	Час (модифікований метод)
2000	0.12	0.07	2.31s	2.49s
4000	0.16	0.16	2.34s	2.48s
6000	0.21	0.26	2.33s	2.48s

8000	0.38	0.49	2.41s	2.63s
10000	0.54	0.67	5.62s	6.1s

Продовження табл. 4.1

Довжина тексту та відсоток схожості	ланцюг Маркова	Модифікований ланцюг Маркова	Час (базовий метод)	Час (модифікований метод)
12000	0.65	0.76	5.7s	6.13s
14000	0.67	0.81	5.69s	6.24s
16000	0.83	0.92	6.1s	7.3s
18000	0.90	0.95	15.3s	17.24s
20000	0.91	0.98	15.6s	17.28s
22000	0.89	0.96	15.6s	17.96s
24000	0.93	0.97	16.1s	18.67s

Відповідно до даних, що наведені у табл. 4.1, можна побачити, що модифікований метод автоматизованого визначення авторства текстів, що базується на ланцюгу Маркова починає давати точніші результати при кількості символів 6000.

Це пов’язано з тим, що для модифікованого методу були застосовані нові характеристики, на яких засновано аналіз і пошук інваріанту. Частотні характеристики тексту на основі триграм та біграм тексту а також частота вживання слів та службових частин мови виявились більш ефективними

характеристиками, що показують авторський стиль, аніж використання для аналізу лише однієї характеристики, як це було в оригінальному методі.

Оскільки у модифікованому методі для обчислення авторського інваріанту було обрано 4 характеристики, це дозволяє більш точно обчислити цей інваріант. Кожна характеристика аналізує текст на різні прикмети. Зокрема, частота вживання слів дозволяє судити про певні особливості тексту, такі, як час, у який він був написаний, освітній рівень автора, тощо. Частота вживання слів – більш несвідомий параметр, але емпіричним шляхом було встановлено, що він досить точно характеризує авторський стиль. Частота біграм та триграм – найбільш ефективні частотні характеристики для аналізу тексту, оскільки біграми застосовується у стандартному методі, а триграми дозволяють аналізувати також окремі слова, які складаються з 3 букв, що дозволяє скласти більш точну матрицю переходів від однієї букви до іншої. За рахунок більш повного охоплення стилістики автора, модифікований метод показує вищу точність.

Звісно, такі обчислення потребують більше часу, за рахунок обчислення частотних характеристик та попередньої обробки тексту, але, як показав аналіз, програш у часі не є настільки значним.

До 6 тисяч символів результати обох методів коливаються. Це пов’язане з тим, що методи мали надто мало символів для виявлення статистичних залежностей, що є показником стійкості методу. Обидва методи не є абсолютно стійкими, оскільки основні концепти, які використовуються в них однакові.

За результатами вищенаведеного аналізу можемо зробити висновок, що модифікований метод визначення тексту на основі ланцюгів Маркова точніший за оригінальний. Також аналіз доводить, що час, необхідний модифікованому методу на аналіз, збільшився на прийнятну величину.

4.5. Висновки

У даному розділі було наведено критерій оцінки ефективності модифікованого методу автоматизованого визначення авторства текстів, основаного на ланцюгу Маркова. Обґрунтовано вибір даних, що використовувались для аналізу ефективності, та проведено порівняльний аналіз ефективності оригінального та модифікованого методів.

Також, були наведені конструкції мови C# та еквівалентні їм базові конструкції мови.

За результатами порівняння можна зробити наступні висновки:

- модифікація методу дозволяє отримати кращий результат за критерієм точності;
- при цьому алгоритм значно виграє за критерієм швидкодії по часу;

У подальших дослідженнях можна проаналізувати стійкість методу до незначних змін.

ВИСНОВКИ

Проведено огляд методів, які використовуються для визначення авторства тексту. На основі огляду було встановлено критерії, за якими можна проводити їх порівняння та визначено основний критерій, який необхідно поліпшити.

Розглянуто існуючі програмні засоби, які вирішують задачу пошуку автора та використовують розглянуті методи. Було встановлено, що найефективнішим методом є метод ланцюгів Маркова, тож він і був обраний за основу нового методу.

Для покращення точності визначення авторства тексту пропонується модифікований метод на основі ланцюгів Маркова, який використовує більше характеристик для визначення авторського інваріанту а також нові підходи до попередньої обробки текстових даних.

Розглянуто методи попередньої обробки текстових даних, та проведено огляд характеристик для формування авторського інваріанту, на яких ґрунтується подальший аналіз у модифікованому методі.

Запропонований метод визначення авторства тексту на основі ланцюгів Маркова складається з таких етапів: попереднє оброблення текстових даних, аналіз даних після попередньої обробки за допомогою ланцюгів Маркова, проведення порівняння між частотними характеристиками вхідного тексту та авторськими інваріантами, зміна інваріанту за допомогою нового тексту.

Також у рамках даної роботи розроблено програмне забезпечення, яке реалізує модифікований метод, та може використовуватись для визначення авторства тексту.

Проведено аналіз методів та застосунків, які використовуються для проектування та розробки програмних систем і, відповідно, обрано засоби для розробки вищезазначеного програмного забезпечення.

Розроблений застосунок відповідає всім висунутим вимогам до програмного забезпечення, таким, як:

– функціональна повнота;
– зручність;
– надійність;
– технічна ефективність;
– адаптивність;
– вартість.

Наведено приклади обраних тестових даних для розробленого застосунку та показано, яких чином дані змінюються під час попередньої обробки.

Проведено аналіз ефективності модифікованого методу визначення авторства тексту у порівнянні з оригінальним методом, та виявлено, що за рахунок обраних характеристик для визначення авторського інваріанту, а саме, таких частотних характеристик, як використання біграм, триграм, частота вживання слів та частота вживання службових слів, та методам попередньої обробки текстових даних, модифікований метод дає збільшення точності аналізу з порівняно невеликим програшом у часі.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Романов А.С. Методика и программный комплекс для идентификации автора неизвестного текста: Автореф. дис. канд. техн. наук. Томск, 2010. 26 с.
2. Рогов А.А., Гурин Г.Б., Котов А.А., Сидоров Ю.В., Суровцова Т.Г. Программный комплекс СМАЛТ // Электронные библиотеки: перспективные методы и технологии, электронные коллекции: Труды X Всерос. науч. конф. «RCDL'2008». Дубна, 2008. С. 155–160.
3. Марков А.А. Об одном применении статистического метода // Известия Императорской Академии наук. Сер. 6. 1916. Т. 10, № 4. С. 239–242.
4. Айвазян С.А., Бухштабер В.М., Енюков И.С., Мешалкин Л.Д. Прикладная статистика. Классификация и снижение размерности. — М.: Финансы и статистика, 1989. — 607 с.
5. Рао С.Р., Линейные статистические методы и их применения. — М.: Наука (Физматлит), 1968. — 548 с.
6. Jolliffe I.T. Principal Component Analysis, Series: Springer Series in Statistics, 2nd ed., Springer, NY, 2002, XXIX, 487 p. 28 illus. ISBN 978-0-387-95442-4
7. Айвазян С.А., Бухштабер В.М., Енюков И.С., Мешалкин Л.Д. Прикладная статистика. Классификация и снижение размерности. — М.: Финансы и статистика, 1989. — 607 с.
8. Рао С.Р., Линейные статистические методы и их применения. — М.: Наука (Физматлит), 1968. — 548 с.
9. Jolliffe I.T. Principal Component Analysis, Series: Springer Series in Statistics, 2nd ed., Springer, NY, 2002, XXIX, 487 p. 28 illus. ISBN 978-0-387-95442-4

10. Statsoft. Электронный учебник по статистике. [Електронний ресурс] — Режим доступу: http://statsoft.ru/home/textbook/modules/stcluan.html
11. Kolmogoroff A.N. Sulla determinazione empirica di una legge di distribuzione // Giornale dell` Istituto Italiano degly Attuari. 1933. — Vol. 4. — № 1. — P. 83-91.
12. Большев Л.Н., Смирнов Н.В. Таблицы математической статитики. М.: Наука, 1983.
13. Лемешко Б.Ю., Постовалов С.Н. О зависимости предельных распределений статистик X^2 Пирсона и отношения правдоподобия от способа группирования данных // Заводская лаборатория. 1998. Т. 64. — № 5. — С. 56-63.
14. Никулин М.С. Критерий хи-квадрат для непрерывных распределений с параметрами сдвига и масштаба // Теория вероятностей и её применение. — 1973. — Т. XVIII, № 3. — С. 583 — 591.
15. Дискретная математика: алгоритмы [Електронний ресурс] — Режим доступу: http://rain.ifmo.ru/cat/view.php/theory/processes-automata/markov-2008
16. Беляев А., Гаврилов М., Масальских А., Медвинский М. Марковские процессы, 2004.
17. Романовский И.В. Дискретный анализ: Учебное пособие для студентов, 3-е изд. — СПб: Невский Диалект; БХВ Петербург, 2003
18. Хмелёв Д.В. Распознавание автора текста с использованием цепей А.А. Маркова // Вестн. МГУ. Сер. 9: Филология. 2000. № 2. С. 115–126.
19. Хмелёв Д.В. Классификация и разметка текстов с использованием методов сжатия данных // Все о сжатии данных, изображений и видео. 2003. [Електронний ресурс] — Режим доступу: http://compression.ru/download/articles/classif/intro.html

20. Программные продукты и системы [Електронний ресурс] — Режим доступу http://www.swsys.ru/index.php?page=article&id=3703
21. Севбо И.П. Графическое представление синтаксических структур и стилистическая диагностика. Киев: Наук. дум., 1981. — 192 с.
22. Мартыненко Г.Я. Основы стилеметрии. Л.: ЛГУ, 1988. — 170 с.
23. Рогов А.А., Сидоров Ю. +В., Король А.В. Автоматизированная система обработки и анализа литературных текстов СМАЛТ // Труды и материалы II Междунар. конгресса исследователей русского языка «Русский язык: исторические судьбы и современность». М: МГУ, 2004. С. 485–486.
24. Шевелёв О.Г. Разработка и исследование алгоритмов сравнения стилей текстовых произведений: Автореф. дис. канд. техн. наук. Томск, 2006. 18 с.
25. Шевелёв О.Г. Методы автоматической классификации текстов на естественном языке: Учеб. пособие. Томск: ТМЛ-Пресс, 2007. — 144 с.
26. Романов А.С., Мещеряков Р.В. Идентификация автора текста с помощью аппарата опорных векторов // Компьютерная лингвистика и интеллектуальные технологии: по материалам ежегодной Международной конференции «Диалог-2009». М.: РГГУ, 2009. Вып. 8,
№ 15. С. 432–437.
27. Морозов Н.А. Лингвистические спектры: средство для отличения плагиатов от истинных произведений того или иного известного автора. Стилеметрический этюд. //Известия отд. русского языка и словестности Имп.Акад.наук, Т.XX, кн.4, 1915.
28. Марков А.А. Об одном применении статистического метода. // Известия Имп.Акад.наук, серия VI, Т.X, N4, 1916, с.239.

29. Марков А.А. Пример статистического исследования над текстом “Евгения Онегина” иллюстрирующий связь испытаний в цепь. // Известия Имп.Акад.наук, серия VI, Т.X, N3, 1913, с.153.
30. Распознавание автора текста с использованием цепей А.А. Маркова. Д.В. Хмелев [Електронний ресурс] — Режим доступу http://www.philol.msu.ru/~lex/khmelev/published/vestnik/a.pdf
31. Кукушкина О.В., Поликарпов А.А., Хмелѐв Д.В.. Определение авторства текста с использованием буквенной и грамматическойинформации. Проблемы передачи информации, 37(2): с. 96-108, 2001

ДОДАТКИ

Додаток 1

Копія презентації

Модифікований метод визначення авторства тексту на основі ланцюгів Маркова

Виконав

Студент 6 курсу групи КП-71мн

Замекула Олексій Ігорович

Керівник

доцент кафедри ПЗКС, к.т.н.

Заболотня Тетяна Миколаївна

Мета роботи

- Мета дослідження полягає у підвищені точності визначення авторства тексту шляхом розроблення та реалізації модифікованого методу аналізу текстів на основі методу ланцюгів Маркова та відповідних програмних засобів.

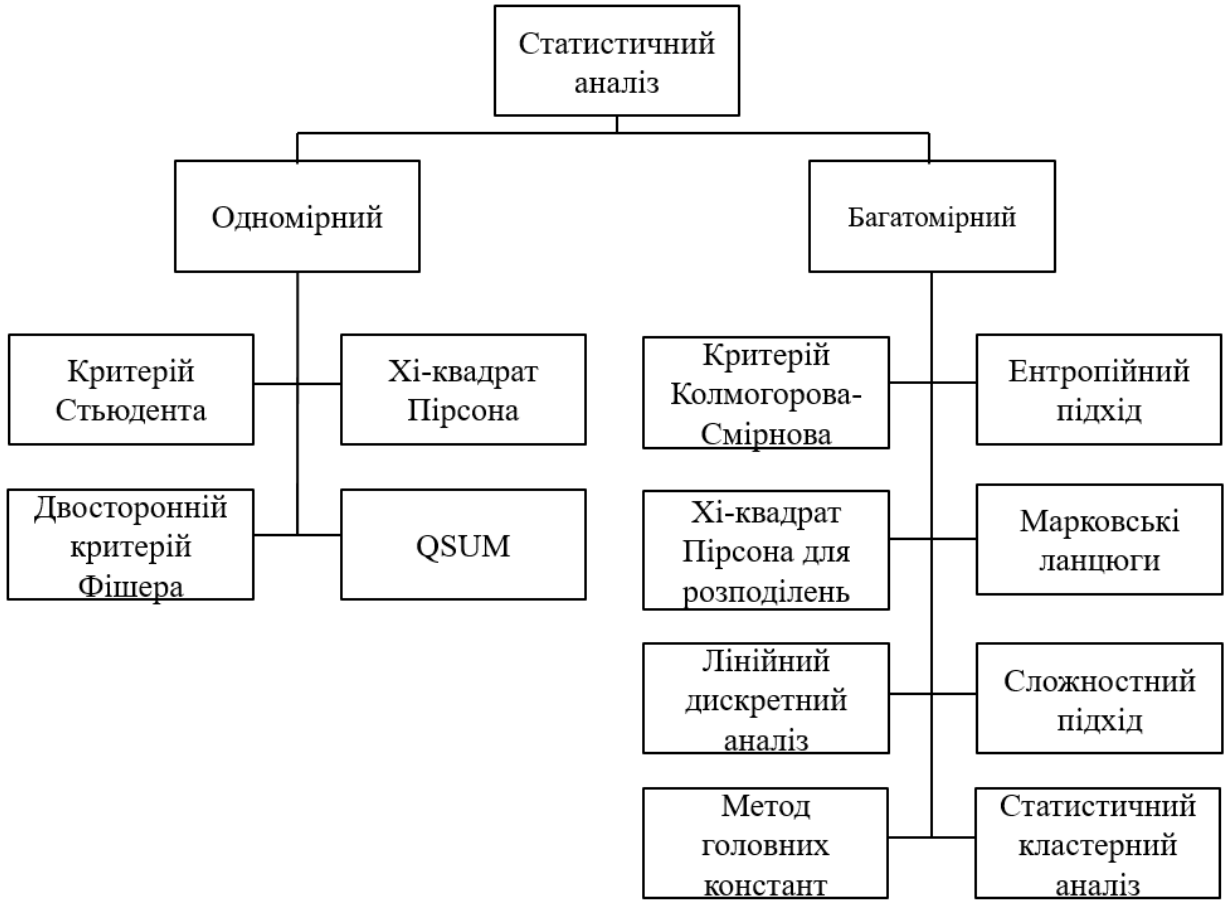
Об’єкт і предмет дослідження

- Об’єктом дослідження в даній роботі є процеси автоматизованого аналізу текстових даних за різними ознаками, що характеризують авторський стиль, зокрема, буквосполучення та використання службових частин мови.
- Предметом дослідження є методи, способи та алгоритми автоматизованого визначення авторства тексту.

Визначення

- Авторський інваріант - це кількісна характеристика літературних текстів чи якийсь параметр, який однозначно характеризує своєю поведінкою твори одного автора або невеликого числа «близьких авторів», і приймає істотно різні значення для творів різних груп авторів. Авторський інваріант застосовується в задачі ідентифікації авторства тексту.

Методи визначення авторства



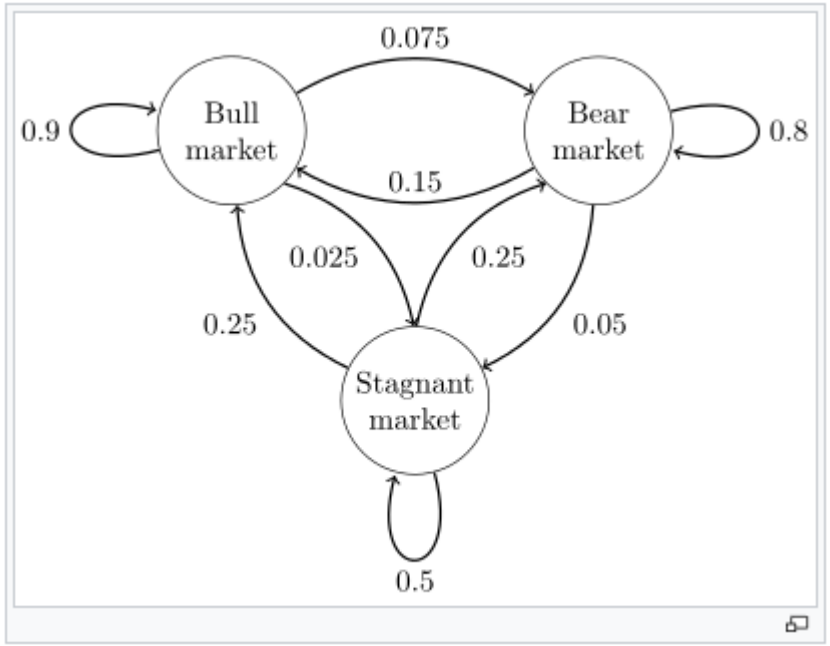
Порівняння існуючих програмних засобів

Назва	Метод	Зміна параметрів методу	Необхідний мінімальний об’єм тексту	Точність, %
Лінгвоаналізатор	Марківські ланцюги, відносна ентропія	Ні	40000 символів	84-89
Атрибутор	Маркавські ланцюги	Ні	20000 символів	80-85
СМАЛТ	Критерії Стьюдента, Колмогорова, кластерний аналіз	Ні	500 слів	невідомо
Стилеаналізатор	Марківські ланцюги, нейронні мережі	Так	30000 символів	90-98
Авторовед	Нейронні мережі, метод опорних векторів, QSUM	Так	20000 символів	95-98

Метод ланцюгів Маркова

Діаграма стану для простого прикладу показана на малюнку праворуч, використовуючи орієнтований граф для зображення переходів стану.

$$P = \begin{bmatrix} 0.9 & 0.075 & 0.025 \\ 0.15 & 0.8 & 0.05 \\ 0.25 & 0.25 & 0.5 \end{bmatrix}.$$



Методи попередньої обробки текстів

- Нормалізація тексту
- Вилучення службових частин мови
- Приведення форм слів
- Вилучення пропусків

Нормалізація тексту

Алгоритм попередньої обробки складається з таких етапів:

- Видалення прямої мови
- Вилучення слів, що починаються з великих літер
- Вилучення розділових знаків

Вилучення службових частин мови

Вилучаються всі слова, які є службовими частинами мови:

- Прийменники
- Сполучники
- Частки

Перелік відповідних слів береться з словника.

Приведення форм слів

Наступний етап полягає у приведені слів до початкової форми. Оскільки у кожного слова є чимала кількість різних форм (відмінки, множина, звороти і т.д.), для статистичного аналізу ці характеристики не є суттєвою інформацією. Після цього етапу можна побудувати частотні характеристики тексту на основі вживання різних слів. При цьому слова «буква», «букви», «буквою» будуть вважатися одним і тим самим словом - «буква».

Вилучення пропусків

Для аналізу частотних характеристик на основі вживання n-грам буквосполучень та переходу з однієї букви до іншої, як випадок реалізації ланцюга Маркова необхідно прибрати всі пропуски. Отже, на виході ми отримаємо послідовність літер українського алфавіту, які можна аналізувати.

Властивості характеристики авторського інваріанту

Основні властивості, якими повинна володіти числова характеристика авторського інваріанта:

- Вона повинна бути досить «масова», інтегральна, щоб слабко контролюватися автором на свідомому рівні.
- Шуканий параметр повинен зберігати «постійне значення» для творів даного автора.
- Параметр повинен впевнено розрізняти між собою різні групи письменників.

Порівняння результатів аналізу для різних характеристик

Випадок (а)			**Випадок (б)**		
Словоформи тексту			Приведені форми		
R	Пари букв	Одиночні	R	Пари букв	Одиночні
0	282/385	27/385	0	240/385	12/385
1	21/385	55/385	1	29/385	40/385
2	9/385	25/385	2	17/385	19/385
3	5/385	24/385	3	9/385	16/385
4	5/385	17/385	4	6/385	16/385
≥ 5	63/385	237/385	≥5	84/385	282/385
M	3,38	12,69	M	4,77	17,88

Порівняння результатів аналізу для різних характеристик

Випадок (в)			**Випадок (г)**		
Узагальнені граматичні класи			«Повні» граматичні класи		
R	Парні	Одиночні	R	Парні	Одиночні
0	235/385	128/385	0	15/385	6/385
1	31/385	43/385	1	21/385	12/385
2	16/385	29/385	2	9/385	6/385
3	8/385	15/385	3	12/385	6/385
4	11/385	17/385	4	19/385	8/385
≥5	84/385	153/385	≥5	309/385	347/385
M	5,43	10,13	M	17,76	31,93

Модифікований метод

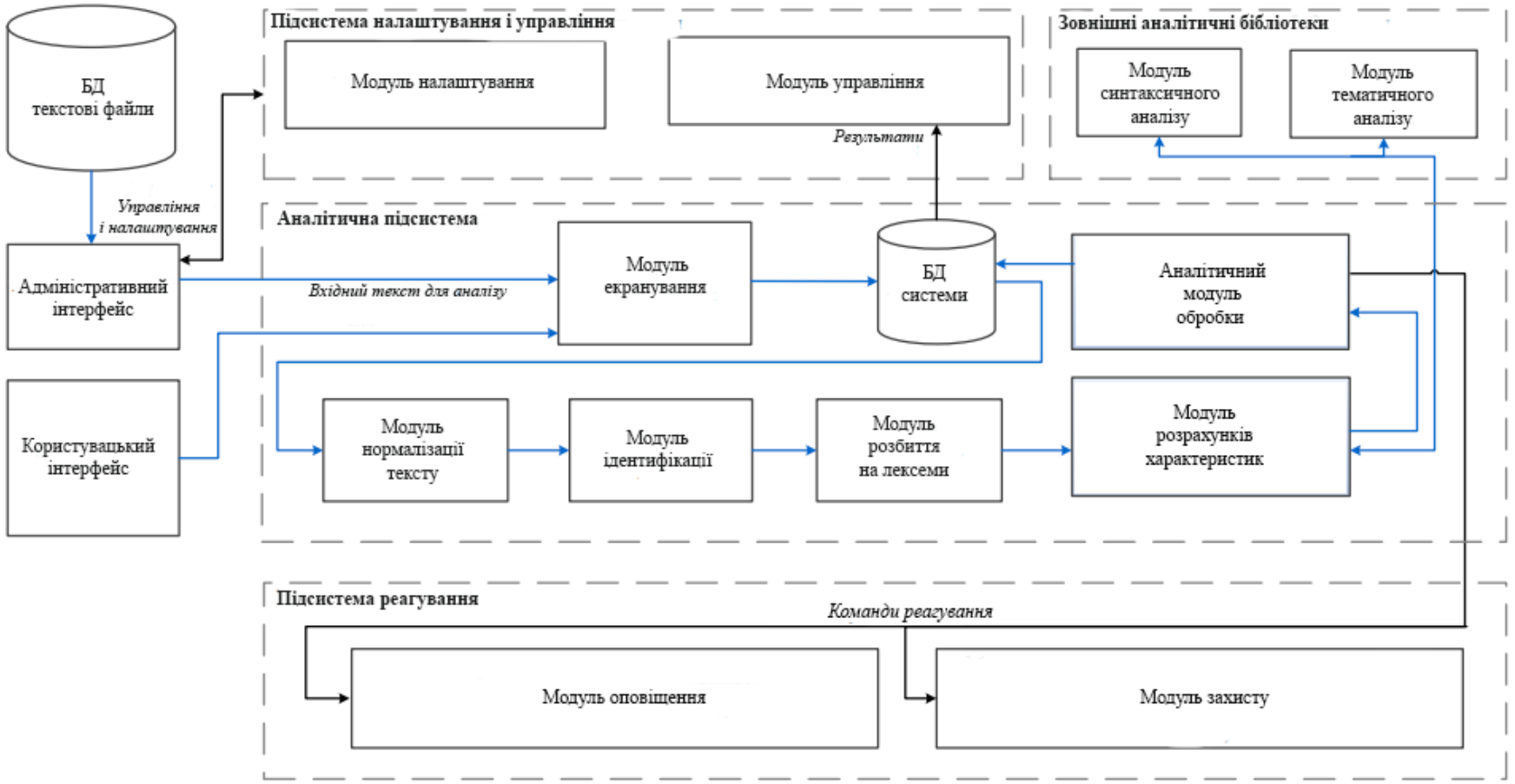
Запропонований метод визначення авторства на основі ланцюгів Маркова складається з наступних етапів:

- Попередня обробка тексту
- Аналіз отриманого результату методом ланцюгів Маркова;
- Оцінка ступеня схожості заданого тексту з авторськими інваріантами;
- За умови однозначної відповіді (ймовірність співпадіння з одним із інваріантів вища 95%, співпадіння з іншими менша 50%) перерахунок відповідного інваріанту.

Засоби проектування та розробки

- Rational Rose
- .NET Standart 2.1
- .NET Framework 4.7.2
- Visual Studio 2017
- MSSQL Server 2012

Архітектура системи



Аналіз роботи модифікованого методу

Довжина тексту та відсоток схожості	ланцюг Маркова	Модифікований ланцюг Маркова	Час (базовий метод)	Час (модифікований метод)
2000	0.12	0.07	2.31s	2.49s
4000	0.31	0.36	2.34s	2.48s
6000	0.63	0.52	2.33s	2.48s
8000	0.77	0.89	2.41s	2.63s
10000	0.14	0.08	5.62s	6.1s
12000	0.3	0.34	5.7s	6.13s
14000	0.67	0.56	5.69s	6.24s
16000	0.83	0.91	6.1s	7.3s
18000	0.09	0.03	15.3s	17.24s
20000	0.29	0.25	15.6s	17.28s
22000	0.7	0.56	15.6s	17.96s
24000	0.9	0.95	16.1s	18.67s

Наукова новизна

- Запропоновано модифікований метод автоматизованого визначення авторства текстів на основі методу ланцюгів Маркова, який відрізняється від існуючих класичних методів підвищеною точністю визначення автора за рахунок використання методу пошуку авторського інваріанту на основі частотних характеристик тексту.

Шляхи подальшого дослідження

- Подальший пошук характеристик, які дозволять більш точно виділяти інваріант
- Підвищення швидкості попередньої обробки

Шляхи подальшого дослідження

- Подальший пошук характеристик, які дозволять більш точно виділяти інваріант
- Підвищення швидкості попередньої обробки

Публікації

- Наукова конференція магістрантів та аспірантів “Прикладна математика та комп’ютинг” ПМК-2018-2
- Подана публікація в журнал «Інженерія програмного забезпечення»

Висновки

- Розглянуто задачу визначення авторства, виконано огляд існуючих методів та порівняння програмних комплексів, які реалізують ці методи
- Розглянуто основні методи попередньої обробки текстової інформації
- Визначено характеристики, які вказують на авторський інваріант

Висновки

- Запропоновано модифікований метод визначення авторства тексту на основі ланцюгів Маркова
- Розроблено програмне забезпечення, що реалізує оригінальний та запропонований методи
- Було наведено критерії оцінки ефективності на порівняно роботу методів за цими критеріям
- Запропоновано шляхи подальшого дослідження

Дякую за увагу!

Додаток 2

Лістинг тексту програми

Лістинг 1. Клас для визначення поточного стану ланцюга Маркова.

```
namespace Markov
{
    using System;
    using System.Collections;
    using System.Collections.Generic;
    using System.Linq;

    /// <summary>
    /// Represents a state in a Markov chain.
    /// </summary>
    /// <typeparam name="T">The type of the constituent parts of each state in
the Markov chain.</typeparam>
    public class ChainState<T> : IEquatable<ChainState<T>>, IList<T>
    {
        private readonly T[] items;

        /// <summary>
        /// Initializes a new instance of the <see cref="ChainState{T}"/>
class with the specified items.
        /// </summary>
        /// <param name="items">An <see cref="IEnumerable{T}"/> of items to be
copied as a single state.</param>
        public ChainState(IEnumerable<T> items)
        {
            if (items == null)
            {
                throw new ArgumentNullException(nameof(items));
            }

            this.items = items.ToArray();
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="ChainState{T}"/>
class with the specified items.
        /// </summary>
        /// <param name="items">An array of <typeparamref name="T"/> items to
be copied as a single state.</param>
        public ChainState(params T[] items)
        {
            if (items == null)
            {
                throw new ArgumentNullException(nameof(items));
            }

            this.items = new T[items.Length];
            Array.Copy(items, this.items, items.Length);
        }

        /// <inheritdoc />
        public int Count => this.items.Length;

        /// <inheritdoc />
        public bool IsReadOnly => true;

        /// <inheritdoc />
        public T this[int index]
        {
```

```
            get { return this.items[index]; }
            set { throw new NotSupportedException(); }
        }

        /// <summary>
        /// Determines whether two specified instances of <see
cref="ChainState{T}"/> are not equal.
        /// </summary>
        /// <param name="a">The first <see cref="ChainState{T}"/> to
compare.</param>
        /// <param name="b">The second <see cref="ChainState{T}"/> to
compare.</param>
        /// <returns>true if <paramref name="a"/> and <paramref name="b"/> do
not represent the same state; otherwise, false.</returns>
        public static bool operator !=(ChainState<T> a, ChainState<T> b)
        {
            return !(a == b);
        }

        /// <summary>
        /// Determines whether two specified instances of <see
cref="ChainState{T}"/> are equal.
        /// </summary>
        /// <param name="a">The first <see cref="ChainState{T}"/> to
compare.</param>
        /// <param name="b">The second <see cref="ChainState{T}"/> to
compare.</param>
        /// <returns>true if <paramref name="a"/> and <paramref name="b"/>
represent the same state; otherwise, false.</returns>
        public static bool operator ==(ChainState<T> a, ChainState<T> b)
        {
            if (object.ReferenceEquals(a, b))
            {
                return true;
            }
            else if (a is null || b is null)
            {
                return false;
            }

            return a.Equals(b);
        }

        /// <inheritdoc />
        public void Add(T item) => throw new NotSupportedException();

        /// <inheritdoc />
        public void Clear() => throw new NotSupportedException();

        /// <inheritdoc />
        public bool Contains(T item) => ((IList<T>)this.items).Contains(item);

        /// <inheritdoc />
        public void CopyTo(T[] array, int arrayIndex) =>
this.items.CopyTo(array, arrayIndex);

        /// <inheritdoc />
        public override bool Equals(object obj)
        {
```

```
            if (obj is ChainState<T> chain)
            {
                return this.Equals(chain);
            }

            return false;
        }

        /// <summary>
        /// Indicates whether the current object is equal to another object of
the same type.
        /// </summary>
        /// <param name="other">An object to compare with this object.</param>
        /// <returns>true if the current object is equal to the <paramref
name="other"/> parameter; otherwise, false.</returns>
        public bool Equals(ChainState<T> other)
        {
            if (other is null)
            {
                return false;
            }

            if (this.items.Length != other.items.Length)
            {
                return false;
            }

            for (var i = 0; i < this.items.Length; i++)
            {
                if (!this.items[i].Equals(other.items[i]))
                {
                    return false;
                }
            }

            return true;
        }

        /// <inheritdoc />
        public IEnumerator<T> GetEnumerator() =>
((IList<T>)this.items).GetEnumerator();

        /// <inheritdoc />
        IEnumerator IEnumerable.GetEnumerator() => this.GetEnumerator();

        /// <inheritdoc />
        public override int GetHashCode()
        {
            var code = this.items.Length;

            for (var i = 0; i < this.items.Length; i++)
            {
                code = (code * 37) + this.items[i].GetHashCode();
            }

            return code;
        }

        /// <inheritdoc />
```

```
        public int IndexOf(T item) => throw new NotSupportedException();

        /// <inheritdoc />
        public void Insert(int index, T item) => throw new
NotSupportedException();

        /// <inheritdoc />
        public bool Remove(T item) => throw new NotSupportedException();

        /// <inheritdoc />
        public void RemoveAt(int index) => throw new NotSupportedException();
    }
}
```

Лістинг 2. Реалізація ланцюга Маркова.

```
namespace Markov
{
    using System;
    using System.Collections.Generic;
    using System.Linq;

    /// <summary>
    /// Builds and walks interconnected states based on a weighted
probability.
    /// </summary>
    /// <typeparam name="T">The type of the constituent parts of each state in
the Markov chain.</typeparam>
    public class MarkovChain<T>
        where T : IEquatable<T>
    {
        private readonly Dictionary<ChainState<T>, Dictionary<T, int>> items =
new Dictionary<ChainState<T>, Dictionary<T, int>>();
        private readonly int order;
        private readonly Dictionary<ChainState<T>, int> terminals = new
Dictionary<ChainState<T>, int>();

        /// <summary>
        /// Initializes a new instance of the <see cref="MarkovChain{T}"/>
class.
        /// </summary>
        /// <param name="order">Indicates the desired order of the <see
cref="MarkovChain{T}"/>.</param>
        /// <remarks>
        /// <para>The <paramref name="order"/> of a generator indicates the
depth of its internal state.  A generator
        /// with an order of 1 will choose items based on the previous item, a
generator with an order of 2
        /// will choose items based on the previous 2 items, and so on.</para>
        /// <para>Zero is not classically a valid order value, but it is
allowed here.  Choosing a zero value has the
        /// effect that every state is equivalent to the starting state, and
so items will be chosen based on their
        /// total frequency.</para>
        /// </remarks>
        public MarkovChain(int order)
        {
            if (order < 0)
            {
```

```
                throw new ArgumentOutOfRangeException(nameof(order));
            }

            this.order = order;
        }

        /// <summary>
        /// Adds the items to the generator with a weight of one.
        /// </summary>
        /// <param name="items">The items to add to the generator.</param>
        public void Add(IEnumerable<T> items) => this.Add(items, 1);

        /// <summary>
        /// Adds the items to the generator with the weight specified.
        /// </summary>
        /// <param name="items">The items to add to the generator.</param>
        /// <param name="weight">The weight at which to add the items.</param>
        public void Add(IEnumerable<T> items, int weight)
        {
            if (items == null)
            {
                throw new ArgumentNullException(nameof(items));
            }

            var previous = new Queue<T>();
            foreach (var item in items)
            {
                var key = new ChainState<T>(previous);

                this.Add(key, item, weight);

                previous.Enqueue(item);
                if (previous.Count > this.order)
                {
                    previous.Dequeue();
                }
            }

            var terminalKey = new ChainState<T>(previous);
            var newWeight = Math.Max(0,
this.terminals.ContainsKey(terminalKey)
                ? weight + this.terminals[terminalKey]
                : weight);
            if (newWeight == 0)
            {
                this.terminals.Remove(terminalKey);
            }
            else
            {
                this.terminals[terminalKey] = newWeight;
            }
        }

        /// <summary>
        /// Adds the item to the generator, with the specified items preceding
it.
        /// </summary>
        /// <param name="previous">The items preceding the item.</param>
        /// <param name="item">The item to add.</param>
```

```
        /// <remarks>
        /// See <see cref="MarkovChain{T}.Add(IEnumerable{T}, T, int)"/> for
remarks.
        /// </remarks>
        public void Add(IEnumerable<T> previous, T item)
        {
            if (previous == null)
            {
                throw new ArgumentNullException(nameof(previous));
            }

            var state = new Queue<T>(previous);
            while (state.Count > this.order)
            {
                state.Dequeue();
            }

            this.Add(new ChainState<T>(state), item, 1);
        }

        /// <summary>
        /// Adds the item to the generator, with the specified state preceding
it.
        /// </summary>
        /// <param name="state">The state preceding the item.</param>
        /// <param name="next">The item to add.</param>
        /// <remarks>
        /// See <see cref="MarkovChain{T}.Add(ChainState{T}, T, int)"/> for
remarks.
        /// </remarks>
        public void Add(ChainState<T> state, T next) => this.Add(state, next,
1);

        /// <summary>
        /// Adds the item to the generator, with the specified items preceding
it and the specified weight.
        /// </summary>
        /// <param name="previous">The items preceding the item.</param>
        /// <param name="item">The item to add.</param>
        /// <param name="weight">The weight of the item to add.</param>
        /// <remarks>
        /// This method does not add all of the preceding states to the
generator.
        /// Notably, the empty state is not added, unless the <paramref
name="previous"/> parameter is empty.
        /// </remarks>
        public void Add(IEnumerable<T> previous, T item, int weight)
        {
            if (previous == null)
            {
                throw new ArgumentNullException(nameof(previous));
            }

            var state = new Queue<T>(previous);
            while (state.Count > this.order)
            {
                state.Dequeue();
            }
```

```
            this.Add(new ChainState<T>(state), item, weight);
        }

        /// <summary>
        /// Adds the item to the generator, with the specified state preceding
it and the specified weight.
        /// </summary>
        /// <param name="state">The state preceding the item.</param>
        /// <param name="next">The item to add.</param>
        /// <param name="weight">The weight of the item to add.</param>
        /// <remarks>
        /// This adds the state as-is.  The state may not be reachable if, for
example, the
        /// number of items in the state is greater than the order of the
generator, or if the
        /// combination of items is not available in the other states of the
generator.
        ///
        /// A negative weight may be passed, which will have the impact of
reducing the weight
        /// of the specified state transition.  This can therefore be used to
remove items from
        /// the generator. The resulting weight will never be allowed below
zero.
        /// </remarks>
        public void Add(ChainState<T> state, T next, int weight)
        {
            if (state == null)
            {
                throw new ArgumentNullException(nameof(state));
            }

            if (!this.items.TryGetValue(state, out var weights))
            {
                weights = new Dictionary<T, int>();
                this.items.Add(state, weights);
            }

            var newWeight = Math.Max(0, weights.ContainsKey(next)
                ? weight + weights[next]
                : weight);
            if (newWeight == 0)
            {
                weights.Remove(next);
                if (weights.Count == 0)
                {
                    this.items.Remove(state);
                }
            }
            else
            {
                weights[next] = newWeight;
            }
        }

        /// <summary>
        /// Randomly walks the chain.
        /// </summary>
```

```
        /// <returns>An <see cref="IEnumerable{T}"/> of the items
chosen.</returns>
        /// <remarks>Assumes an empty starting state.</remarks>
        public IEnumerable<T> Chain() => this.Chain(Enumerable.Empty<T>(), new
Random());

        /// <summary>
        /// Randomly walks the chain.
        /// </summary>
        /// <param name="previous">The items preceding the first item in the
chain.</param>
        /// <returns>An <see cref="IEnumerable{T}"/> of the items
chosen.</returns>
        public IEnumerable<T> Chain(IEnumerable<T> previous) =>
this.Chain(previous, new Random());

        /// <summary>
        /// Randomly walks the chain.
        /// </summary>
        /// <param name="seed">The seed for the random number generator, used
as the random number source for the chain.</param>
        /// <returns>An <see cref="IEnumerable{T}"/> of the items
chosen.</returns>
        /// <remarks>Assumes an empty starting state.</remarks>
        public IEnumerable<T> Chain(int seed) =>
this.Chain(Enumerable.Empty<T>(), new Random(seed));

        /// <summary>
        /// Randomly walks the chain.
        /// </summary>
        /// <param name="previous">The items preceding the first item in the
chain.</param>
        /// <param name="seed">The seed for the random number generator, used
as the random number source for the chain.</param>
        /// <returns>An <see cref="IEnumerable{T}"/> of the items
chosen.</returns>
        public IEnumerable<T> Chain(IEnumerable<T> previous, int seed) =>
this.Chain(previous, new Random(seed));

        /// <summary>
        /// Randomly walks the chain.
        /// </summary>
        /// <param name="rand">The random number source for the chain.</param>
        /// <returns>An <see cref="IEnumerable{T}"/> of the items
chosen.</returns>
        /// <remarks>Assumes an empty starting state.</remarks>
        public IEnumerable<T> Chain(Random rand) =>
this.Chain(Enumerable.Empty<T>(), rand);

        /// <summary>
        /// Randomly walks the chain.
        /// </summary>
        /// <param name="previous">The items preceding the first item in the
chain.</param>
        /// <param name="rand">The random number source for the chain.</param>
        /// <returns>An <see cref="IEnumerable{T}"/> of the items
chosen.</returns>
        public IEnumerable<T> Chain(IEnumerable<T> previous, Random rand)
        {
```

```
            if (previous == null)
            {
                throw new ArgumentNullException(nameof(previous));
            }
            else if (rand == null)
            {
                throw new ArgumentNullException(nameof(rand));
            }

            var state = new Queue<T>(previous);
            while (true)
            {
                while (state.Count > this.order)
                {
                    state.Dequeue();
                }

                var key = new ChainState<T>(state);

                if (!this.items.TryGetValue(key, out var weights))
                {
                    yield break;
                }

                this.terminals.TryGetValue(key, out var terminalWeight);

                var total = weights.Sum(w => w.Value);
                var value = rand.Next(total + terminalWeight) + 1;

                if (value > total)
                {
                    yield break;
                }

                var currentWeight = 0;
                foreach (var nextItem in weights)
                {
                    currentWeight += nextItem.Value;
                    if (currentWeight >= value)
                    {
                        yield return nextItem.Key;
                        state.Enqueue(nextItem.Key);
                        break;
                    }
                }
            }
        }

        /// <summary>
        /// Gets the items from the generator that follow from an empty state.
        /// </summary>
        /// <returns>A dictionary of the items and their weight.</returns>
        public Dictionary<T, int> GetInitialStates() => this.GetNextStates(new
ChainState<T>(Enumerable.Empty<T>()));

        /// <summary>
        /// Gets the items from the generator that follow from the specified
items preceding it.
        /// </summary>
```

```
        /// <param name="previous">The items preceding the items of
interest.</param>
        /// <returns>A dictionary of the items and their weight.</returns>
        public Dictionary<T, int> GetNextStates(IEnumerable<T> previous)
        {
            var state = new Queue<T>(previous);
            while (state.Count > this.order)
            {
                state.Dequeue();
            }

            return this.GetNextStates(new ChainState<T>(state));
        }

        /// <summary>
        /// Gets the items from the generator that follow from the specified
state preceding it.
        /// </summary>
        /// <param name="state">The state preceding the items of
interest.</param>
        /// <returns>A dictionary of the items and their weight.</returns>
        public Dictionary<T, int> GetNextStates(ChainState<T> state)
        {
            if (this.items.TryGetValue(state, out var weights))
            {
                return new Dictionary<T, int>(weights);
            }

            return null;
        }

        /// <summary>
        /// Gets all of the states that exist in the generator.
        /// </summary>
        /// <returns>An enumerable collection of <see cref="ChainState{T}"/>
containing all of the states in the generator.</returns>
        public IEnumerable<ChainState<T>> GetStates()
        {
            foreach (var state in this.items.Keys)
            {
                yield return state;
            }

            foreach (var state in this.terminals.Keys)
            {
                if (!this.items.ContainsKey(state))
                {
                    yield return state;
                }
            }
        }

        /// <summary>
        /// Gets the weight of termination following from the specified items.
        /// </summary>
        /// <param name="previous">The items preceding the items of
interest.</param>
        /// <returns>A dictionary of the items and their weight.</returns>
        public int GetTerminalWeight(IEnumerable<T> previous)
```

```
        {
            var state = new Queue<T>(previous);
            while (state.Count > this.order)
            {
                state.Dequeue();
            }

            return this.GetTerminalWeight(new ChainState<T>(state));
        }

        /// <summary>
        /// Gets the weights of termination following from the specified
state.
        /// </summary>
        /// <param name="state">The state preceding the items of
interest.</param>
        /// <returns>A dictionary of the items and their weight.</returns>
        public int GetTerminalWeight(ChainState<T> state)
        {
            this.terminals.TryGetValue(state, out var weight);
            return weight;
        }
    }
}
```